

# **INF275-10156 - SEMINARIO DE PRÁCTICA DE INFORMATICA PROYECTO COMPLETO**

**SQDeportes**

**Rolando Andrés Palermo**

DNI: 33.311.173 / Legajo: VINF013533

Profesor: **PABLO ALEJANDRO VIRGOLINI**

**30 de Junio de 2024**

**PROYECTO COMPLETO / INF275-10156 - SEMINARIO DE PRÁCTICA  
DE INFORMATICA**

## Título del proyecto:

Desarrollo de un sistema de gestión para la tienda SQ Deportes, optimizando la experiencia del cliente y mejorando la eficiencia operativa.

## Introducción:

SQ Deportes es una tienda especializada en la venta de ropa y artículos deportivos, comprometida desde hace más de 10 años, en ofrecer productos de calidad y satisfacer las necesidades de sus clientes. Con el fin de mejorar su servicio y aumentar su eficacia y competitividad en el mercado, se propone el desarrollo de un sistema de gestión integral que permita optimizar todas las áreas de la tienda.

Este sistema abordará áreas clave como ventas, control de stock, facturación y arqueo de caja para optimizar sus procesos internos y ofrecer un servicio más ágil y efectivo.

El presente proyecto no solo busca mejorar la eficiencia operativa de SQ Deportes, sino también fortalecer su posición en el mercado y fomentar la fidelización de clientes. De esta forma aspira a establecerse como un referente en la industria minorista de la ciudad de Balcarce, destacándose por su innovación, calidad de servicio y compromiso con la satisfacción del cliente.

## Justificación:

En el contexto actual del competitivo mercado minorista, SQ Deportes reconoce la necesidad imperante de mejorar sus procesos operativos para mantenerse a la vanguardia y satisfacer las demandas cambiantes. La implementación de un sistema de gestión integral no solo es una necesidad estratégica para la empresa, sino que también ofrece una serie de beneficios y oportunidades que justifican su realización:

**Optimización de procesos operativos:** SQ Deportes se enfrenta a desafíos diarios relacionados con la gestión de ventas, control de stock, facturación y arqueo de caja. La implementación de un sistema de gestión permitirá automatizar y agilizar estos procesos, reduciendo el tiempo dedicado a tareas administrativas y optimizando la eficiencia operativa en todas las áreas de la empresa.

**Mejora de la experiencia del cliente:** Al optimizar sus procesos internos, SQ Deportes podrá ofrecer una experiencia de compra más fluida y satisfactoria para sus clientes. Con un mejor control de inventario, tiempos de espera reducidos en cajas y una facturación más precisa, los clientes experimentarán un servicio más rápido, eficiente y personalizado, lo que aumentará su satisfacción y fidelización.

**Toma de decisiones informada:** La implementación de un sistema de gestión proporcionará a SQ Deportes acceso a datos en tiempo real sobre ventas, inventario y rendimiento financiero. Esto permitirá a la empresa tomar decisiones más informadas y estratégicas en áreas clave como planificación de inventario, promociones de ventas y gestión de recursos, lo que contribuirá a su crecimiento y rentabilidad a largo plazo.

**Competitividad en el mercado:** En un entorno minorista altamente competitivo, la capacidad de adaptarse rápidamente a las tendencias del mercado (como descuentos con tarjeta y cobros con

cuenta DNI) y satisfacer las demandas de los clientes es esencial para el éxito. La implementación del sistema posicionará a SQ Deportes como una empresa ágil, eficiente y orientada al cliente, lo que le permitirá competir de manera efectiva en el mercado y diferenciarse de sus competidores.

## Definiciones del proyecto y del sistema:

### Objetivo general del proyecto:

Desarrollar un sistema de gestión integral para la tienda SQ Deportes, que optimice la experiencia del cliente y mejore la eficiencia operativa durante el próximo año.

### Objetivos específicos:

- 1- Implementar un sistema de gestión de ventas según usuario que permita registrar y procesar las transacciones de manera eficiente, reduciendo el tiempo de espera de los clientes en caja y mejorando la precisión en la facturación.
- 2- Establecer un sistema de control de inventario en tiempo real que optimice la gestión de stock, minimizando las pérdidas por agotamiento de productos y asegurando la disponibilidad de productos más demandados.
- 3- Mejorar la eficiencia de la tienda mediante la implementación de un sistema de arqueo de caja diario y cierre de caja mensual, garantizando una gestión transparente de los ingresos y egresos de efectivo y cuenta corriente.

### Diagrama de Gantt

Actividades	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6	Mes 7	Mes 8	Mes 9	Mes 10	Mes 11	Mes 12
Investigación y análisis de requisitos												
Desarrollo del sistema de gestión de ventas												
Implementación del control de inventario												
Desarrollo del sistema de arqueo de caja												
Pruebas y ajustes del sistema												
Capacitación del personal												
Implementación del sistema												

## Definiciones del sistema:

### Objetivo general del sistema:

Desarrollar un sistema de gestión integral para SQ Deportes que recolecte y analice datos para optimizar el control de eficiencia en ventas, gestión de inventario, experiencia del cliente, arqueo de caja mensual y relación con proveedores, facilitando así su gestión y control durante el próximo año.

**Límites del sistema:** El sistema se enfocará en las áreas de ventas, inventario, experiencia del cliente, arqueo de caja mensual y relación con proveedores, dejando fuera aspectos como recursos humanos o contabilidad.

**Alcances del sistema:** El sistema incluirá módulos para registrar ventas por usuario, gestionar inventario, administrar promociones y descuentos, realizar arqueos de caja diarios y mensuales, gestionar proveedores, etc.

**Restricciones del sistema:** Se deberá desarrollar dentro de un presupuesto y un plazo definidos. Además, se deberá integrar con el sistema de punto de venta existente en la tienda.

## Elicitación:

### Proceso de Elicitación:

**Definición del Alcance:** Antes de iniciar la elicitación, se estableció el alcance del proyecto, identificando las áreas clave a abordar, como ventas, inventario, experiencia del cliente, usuarios, arqueo de caja mensual y relación con proveedores.

**Selección de Técnicas:** Se optó por utilizar dos técnicas complementarias: encuestas y entrevistas. Estas técnicas permitirán obtener tanto datos cuantitativos como cualitativos para comprender a fondo las necesidades de los usuarios y partes interesadas.

**Diseño de Encuestas:** Se diseñaron encuestas estructuradas para recopilar datos cuantitativos sobre las preferencias y expectativas de los clientes. Las preguntas se enfocaron en aspectos como la experiencia de compra, la satisfacción con el servicio actual y las sugerencias de mejora.

**Conducta de Entrevistas:** Se planificaron entrevistas semi-estructuradas con empleados de la tienda, gerentes y clientes selectos. Estas entrevistas permitirán explorar en profundidad temas específicos, como desafíos operativos, necesidades no cubiertas y expectativas de un nuevo sistema.

**Personalización de Herramientas:** Se utilizaron herramientas de encuestas en línea para la recolección de datos y herramientas de grabación y análisis de entrevistas para capturar y analizar la información cualitativa.

**Participantes del Proceso:** Participaron empleados de los puntos de venta de la tienda, gerentes, clientes habituales y proveedores clave. Esto garantizó una representación diversa de las perspectivas y necesidades relevantes para el proyecto.

**Análisis de Resultados:** Se analizaron los datos recopilados de encuestas y entrevistas utilizando técnicas de análisis cualitativo y cuantitativo. Se identificaron patrones, tendencias y áreas críticas que deben abordarse en los requerimientos del sistema.

### Actividad del Cliente:

Se espera que los clientes interactúen con el sistema principalmente a través de la interfaz de punto de venta (POS) en la tienda física. Además, se considera a futuro, la posibilidad de desarrollar una aplicación móvil que permita a los clientes realizar compras en línea, ver el inventario disponible y recibir notificaciones sobre promociones y descuentos.

### Tecnologías a Utilizar:

Para el desarrollo del sistema de gestión integral de SQ Deportes, se considerarán las siguientes tecnologías de la información y comunicaciones (TIC):

**Lenguajes de Programación:** Se utilizarán lenguajes como Java para el desarrollo de la aplicación.

**Base de Datos:** Se utilizará una base de datos relacional como MySQL para almacenar información de clientes, productos, transacciones, etc.

**Herramientas de Encuestas y Entrevistas:** Para la recolección y análisis de datos se utilizarán herramientas como Google Forms para las encuestas y software de grabación y análisis de entrevistas como Audacity y NVivo.

### **Análisis de Competencia:**

Se realizará un análisis detallado de los competidores en el mercado minorista de artículos deportivos en la ciudad de Balcarce (Como Open Sport). Se evaluarán aspectos como la variedad de productos, precios, calidad de servicio, experiencia del cliente y estrategias de fidelización. Esto permitirá identificar oportunidades y áreas de mejora para SQ Deportes.

## **Conocimiento del Negocio:**

Al desarrollar un sistema de gestión integral para SQ Deportes, es fundamental tener en cuenta determinados conocimientos del negocio para diseñar soluciones que aborden las necesidades específicas de la empresa. Esto incluye aspectos como la gestión de inventario, la optimización de procesos de venta, la personalización de la experiencia del cliente y la diferenciación frente a la competencia.

Para comprender mejor su funcionamiento y las necesidades específicas de su negocio, es importante tener en cuenta los siguientes aspectos:

**Gama de Productos:** SQ Deportes ofrece una amplia variedad de productos relacionados con el deporte, que van desde ropa deportiva y calzado hasta accesorios y equipos especializados. Es fundamental tener un conocimiento profundo de la gama de productos disponibles, incluyendo marcas, tallas, colores y características técnicas, para garantizar un adecuado control de inventario y satisfacer las necesidades de los clientes.

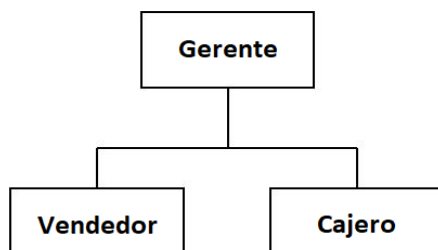
**Ciclo de Temporadas y Tendencias:** El negocio de la moda deportiva está influenciado por el ciclo de temporadas y las tendencias del mercado. Es importante estar al tanto de las últimas tendencias en moda deportiva, así como de los eventos deportivos relevantes que puedan impactar en la demanda de ciertos productos. Esto permitirá a SQ Deportes anticiparse a las necesidades de sus clientes y ofrecer productos actualizados y atractivos.

**Experiencia del Cliente:** La experiencia del cliente juega un papel fundamental en el éxito de SQ Deportes. Desde el momento en que un cliente entra en la tienda hasta que realiza una compra, cada interacción debe ser cuidadosamente gestionada para garantizar la satisfacción del cliente. Esto incluye aspectos como la atención al cliente, la disponibilidad de productos, la comodidad de la tienda y los procesos de pago.

**Competencia en el Mercado:** SQ Deportes opera en un mercado altamente competitivo, donde la diferenciación y la innovación son clave para destacarse. Es importante tener un conocimiento profundo de los competidores en el mercado, incluyendo sus fortalezas, debilidades y estrategias de marketing. Esto permitirá a SQ Deportes identificar oportunidades de mejora y desarrollar estrategias efectivas para mantener su posición competitiva.

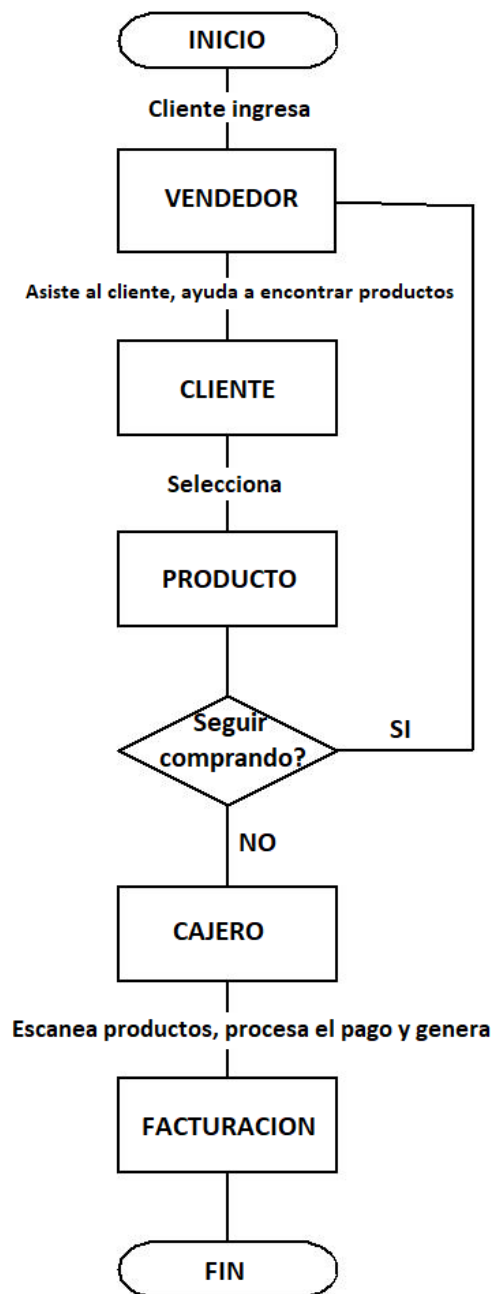
## Diagrama de dominio

Estructura de SQ Deportes:

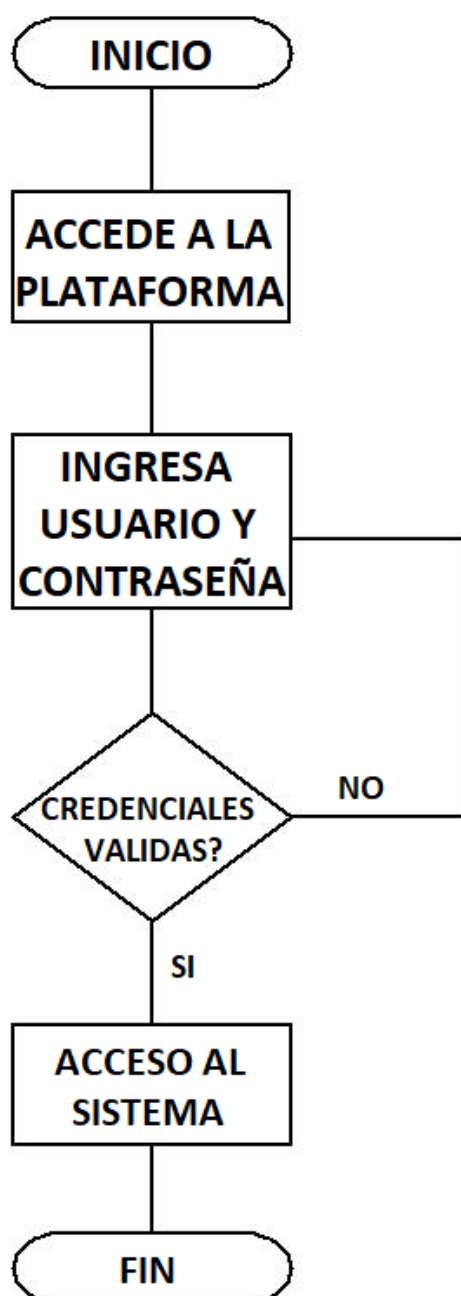


**Aclaración:** La dueña del local cumple el rol de gerencia y dirección.

Proceso:	Gestión de Ventas en Tienda Física
Roles:	Vendedor / Cajero.
Pasos:	<ul style="list-style-type: none"> <li>- El cliente ingresa a la tienda y es recibido por un vendedor.</li> <li>- El vendedor asiste al cliente, ayudándolo a encontrar productos y respondiendo sus preguntas.</li> <li>- Una vez que el cliente ha seleccionado los productos que desea comprar, se dirige a la caja.</li> <li>- El cajero escanea los productos, calcula el total y procesa el pago.</li> <li>- El cliente recibe su recibo de compra y los productos adquiridos.</li> </ul>



Proceso:	Loguin de usuario en el sistema.
Roles:	Cajero
Pasos:	<ul style="list-style-type: none"> <li>- El usuario accede a la plataforma de SQ Deportes.</li> <li>- El sistema muestra la pantalla de inicio de sesión.</li> <li>- El usuario ingresa su nombre de usuario y contraseña.</li> <li>- El sistema valida las credenciales del usuario.</li> <li>- Si las credenciales son válidas, el usuario accede al sistema y puede realizar acciones según su rol.</li> </ul>



Realizar el diagnóstico de los procesos relevados en SQ Deportes (solo se realizaron 2 para no hacer tan extenso el TP)

Proceso:	Gestión de Ventas en Tienda Física
Problema/s:	Tiempos de espera prolongados en caja, inconsistencias en el inventario.
Causa/s:	Falta de personal en caja durante horas pico, falta de sincronización entre el sistema de punto de venta y el inventario físico.



Proceso:	Login de Usuario en Sistema
Problema/s:	Dificultades de acceso al sistema, vulnerabilidades de seguridad.
Causa/s:	Contraseñas olvidadas, problemas de infraestructura de red, falta de políticas de seguridad claras.

## Propuesta de solución:

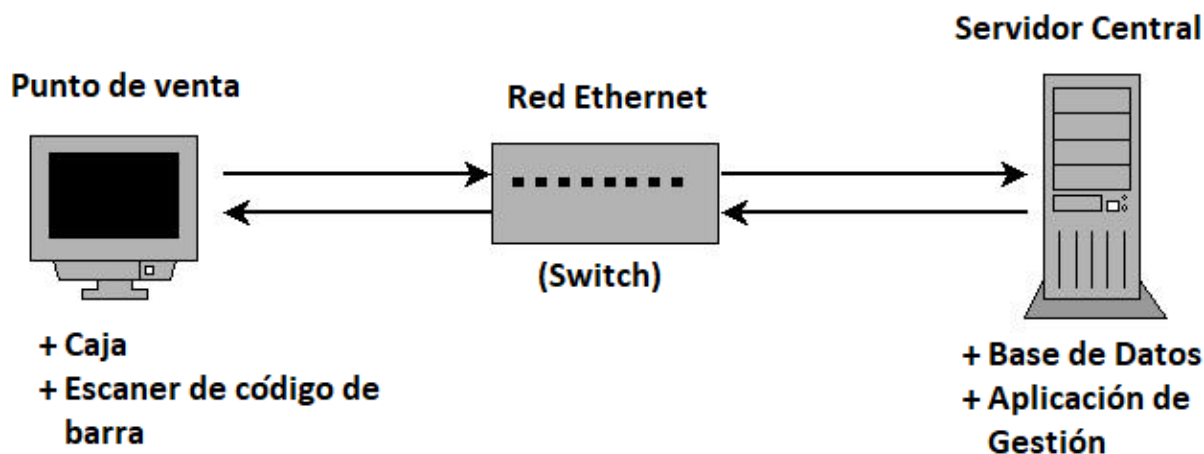
### Propuesta funcional:

Para mejorar la gestión de ventas y el control de inventario en SQ Deportes, se propone desarrollar un sistema de gestión integral que permita registrar y procesar las transacciones de manera eficiente, reduciendo los tiempos de espera en caja y mejorando la precisión en la facturación. El sistema también contemplará un control de inventario en tiempo real para optimizar la gestión de stock y minimizar las pérdidas por agotamiento de productos.

### Propuesta técnica:

El sistema de gestión integral se desarrollará en Java, aprovechando su versatilidad y las bibliotecas disponibles para el escalamiento del sistema. La base de datos se implementará en MySQL, que ofrece un almacenamiento flexible y de alto rendimiento para los datos de ventas e inventario.

Para la comunicación entre los diferentes puntos de venta y la base de datos central, se utilizará una conexión de red local Ethernet. Esto garantizará una comunicación confiable y eficiente entre los diferentes componentes del sistema.



### Descripción de la arquitectura propuesta:

- 1- Punto de Venta: Cada punto de venta en SQ Deportes estará equipado con una caja registradora y un escáner de código de barras para procesar las transacciones de manera eficiente.
- 2- Red Ethernet: La red Ethernet proporcionará la infraestructura de comunicación para conectar los puntos de venta con el servidor central, garantizando una comunicación confiable y segura entre los diferentes componentes del sistema.

- 3- Servidor Central: El servidor central albergará la base de datos central y la aplicación de gestión que procesará y almacenará los datos de ventas e inventario. Esta aplicación también gestionará la comunicación con los puntos de venta a través de la red Ethernet.

Esta arquitectura propuesta proporciona una solución integral para mejorar la gestión de ventas y el control de inventario en SQ Deportes, utilizando tecnologías robustas y una infraestructura de red confiable para garantizar un funcionamiento eficiente del sistema.

## Requerimientos:

**Tabla 1: Requerimientos funcionales**

Requerimiento	Descripción
RFS01	El sistema debe permitir a los empleados iniciar sesión utilizando un nombre de usuario y contraseña únicos.
RFS02	El sistema debe permitir a los administradores agregar, modificar y eliminar cuentas de usuario para los empleados.
RFS03	Los usuarios deben tener roles asignados, como vendedor, cajero o administrador, que determinen sus permisos de acceso dentro del sistema.
RFS04	El sistema debe permitir a los empleados registrar nuevos clientes en la base de datos, incluyendo información como nombre, dirección y datos de contacto.
RFS05	Los empleados deben poder buscar y actualizar la información de los clientes existentes en el sistema.
RFS06	El sistema debe permitir a los empleados agregar nuevos proveedores, incluyendo información como nombre, dirección y datos de contacto.
RFS07	Los empleados deben poder buscar y actualizar la información de los proveedores existentes en el sistema.
RFS08	El sistema debe mantener un registro actualizado del inventario de productos, incluyendo información sobre la cantidad disponible, precios y ubicación.
RFS09	Los empleados deben poder realizar ajustes de inventario, búsquedas, agregar nuevas existencias, eliminar productos obsoletos o registrar pérdidas por daños.
RFS10	El sistema debe permitir a los empleados realizar pedidos de compra, Escanear código de barras, procesar ventas de productos, incluyendo la selección de productos, cálculo de precios, aplicaciones de descuentos y formas de pago.
RFS11	Los empleados deben poder realizar devoluciones de productos y generar reembolsos si es necesario.
RFS12	El sistema debe permitir realizar cobros y generar automáticamente facturas precisas para cada transacción de venta, incluyendo detalles como productos comprados, precios unitarios, impuestos aplicados y total de la venta.
RFS13	Los empleados deben poder imprimir o enviar por correo electrónico las facturas a los clientes al finalizar cada venta.

Estos requerimientos funcionales abordan las necesidades de los usuarios, clientes, proveedores, así como los procesos relacionados con el control de stock, ventas y facturación en el contexto del proyecto de SQ Deportes.

**Tabla 2: Requerimientos no funcionales**

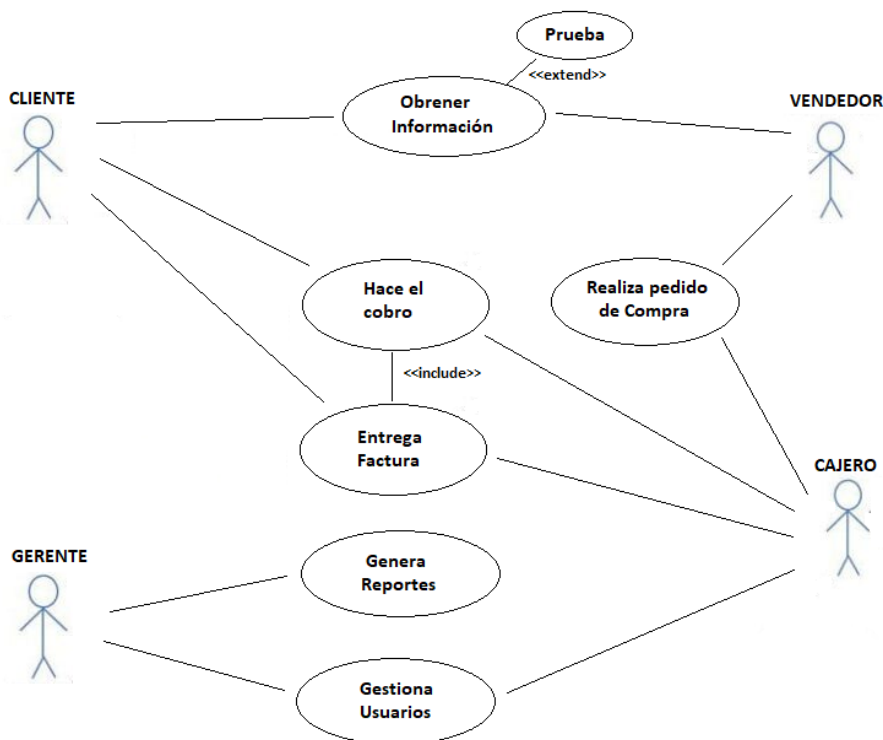
Requerimiento	Descripción
RNF01	El sistema debe estar desarrollado en Java.
RNF02	El sistema debe utilizar una base de datos MySQL para el almacenamiento de datos.
RNF03	El sistema debe tener una disponibilidad de 06hs a 22hs, con un tiempo de inactividad programado mensual para mantenimiento.
RNF04	El sistema debe ser escalable para permitir la adición de nuevas funcionalidades y la expansión a nuevas sucursales en el futuro.
RNF05	El sistema debe estar alojado en una infraestructura propia en las instalaciones de SQ Deportes.
RNF06	El tiempo de respuesta deberá ser menor a los 3 segundos.

**Tabla 3: Requerimientos candidatos**

Para identificar requerimientos candidatos, podríamos considerar aquellos que son fundamentales para el funcionamiento básico del sistema, aquellos que representan una alta prioridad para los stakeholders y aquellos que tienen un impacto significativo en la arquitectura y diseño del sistema. A partir de esta perspectiva, aquí están tres requerimientos que podrían ser clasificados como requerimientos candidatos:

ID Requerimiento	Descripción
RFS01	El sistema debe permitir a los empleados iniciar sesión utilizando un nombre de usuario y contraseña únicos.
RFS10	El sistema debe permitir a los empleados procesar ventas de productos, incluyendo la selección de productos, cálculo de precios, aplicaciones de descuentos y formas de pago.
RNF06	El tiempo de respuesta deberá ser menor a los 3 segundos.

## Inicio del análisis: casos de uso.



#### Identificación de Actores:

- 1- Empleado de Ventas: Representa a los empleados que trabajan en las áreas de ventas y atención al cliente en la tienda. Este actor interactúa directamente con el sistema para realizar operaciones relacionadas con ventas, inventario y atención al cliente.
- 2- Gerente/Administrador: Representa al gerente de la tienda SQ Deportes, quien tiene acceso a funciones administrativas y de supervisión. Este actor puede realizar tareas como la generación de informes, la gestión de usuarios y la configuración del sistema.
- 3- Cliente: Representa a los clientes que interactúan con la tienda SQ Deportes para realizar compras y obtener información sobre productos.
- 4- Cajero: Representa a los empleados encargados de procesar transacciones de venta en las cajas registradoras y entregar la facturación.

#### Trazabilidad

Como referencia, se van a seleccionar los siguientes casos de uso para avanzar:

CU001 Obtener Información

CU002 Realizar Pedido de compra

CU003 Hace el cobro

Requerimiento	Caso de Uso	Actor Principal	Paquete del Análisis	Comentario
RFS10	CU001	Cliente	Cliente obtiene información del producto.	Cliente consulta sobre una remera.
RFS10	CU001	Cliente	Cliente obtiene información del producto.	Indica talla y color de la remera.

RFS09	CU001	Vendedor	Cliente obtiene información del producto.	Busca disponibilidad de stock.
RFS10	CU001	Vendedor/cliente	Cliente obtiene información del producto.	Muestra la remera al cliente.
RFS10	CU002	Vendedor	Realizar Pedido de compra	Registra el pedido de compra de la remera.
RFS10	CU003	Cajero	Hace el cobro	Escanea el código de la remera
RFS11	CU003	Cajero	Hace el cobro	Procesa el pago.
RFS11	CU003	CU003	Hace el cobro	Genera automáticamente la factura.

Tabla CU001:

Caso de uso:	CU001 - Cliente obtiene información del producto
Actores:	Cliente, Vendedor
Referencias:	RFS09, RFS10
Descripción:	Este caso de uso permite al cliente obtener información detallada sobre un producto disponible en la tienda.
Precondición:	-El cliente ha ingresado a la tienda SQ Deportes. -El cliente está interesado en obtener información sobre un producto específico.
Flujo principal:	1 - El cliente busca el producto de interés en la tienda. 2 - El cliente solicita información sobre el producto al Empleado de Ventas. 3 - El Empleado de Ventas busca el producto en el sistema. 4 - El sistema muestra al Empleado de Ventas la información detallada del producto, incluyendo características, precio, disponibilidad, y cualquier otra información relevante. 5 - El Empleado de Ventas proporciona al cliente la información solicitada. 6 - El cliente agradece al Empleado de Ventas y continúa con su decisión de compra.
Postcondición:	El cliente ha recibido la información necesaria sobre el producto y puede tomar una decisión informada sobre su compra.
Flujo alternativo:	Si el producto no está disponible en la tienda, el Empleado de Ventas puede ofrecer alternativas similares al cliente.

Tabla CU002:

Caso de uso:	CU002 - Vendedor genera pedido de compra
Actores:	Vendedor
Referencias:	RFS10
Descripción:	Este caso de uso permite al vendedor de la tienda SQ Deportes generar un pedido de compra el cual será entregado al cajero.
Precondición:	-El vendedor ha iniciado sesión en el sistema.
Flujo principal:	1 - El vendedor selecciona la opción de "Generar Pedido de Compra" en la interfaz del sistema.

	<p>2 - El sistema muestra un formulario para que el vendedor complete los detalles del pedido de compra, incluyendo los productos a ordenar, cantidades y proveedores.</p> <p>3 - El vendedor completa el formulario con la información requerida.</p> <p>4 - El sistema valida la información ingresada por el vendedor.</p> <p>5 - El sistema registra el pedido de compra en la base de datos y genera una confirmación.</p>
Postcondición:	Se ha generado exitosamente un pedido de compra en el sistema, listo para ser entregado al cajero para su procesamiento.
Flujo alternativo:	- Si la información ingresada por el vendedor es incorrecta o incompleta, el sistema muestra un mensaje de error y permite al vendedor corregir los datos.

Tabla CU003:

Caso de Uso:	CU003 - Cajero hace el cobro
Actores:	Cajero
Referencias:	RFS10, RFS11
Descripción:	Este caso de uso permite al cajero de la tienda SQ Deportes realizar el cobro de los productos comprados por el cliente.
Precondición:	- El cajero ha iniciado sesión en el sistema.
Flujo principal:	<p>1 - El cajero escanea el código de barras de los productos adquiridos por el cliente.</p> <p>2 - El sistema muestra el detalle de la compra, incluyendo los productos, precios y total a pagar.</p> <p>3 - El cajero confirma el monto total con el cliente y procesa el pago utilizando el método de pago seleccionado por el cliente.</p> <p>4 - El sistema registra la transacción y genera automáticamente la factura correspondiente.</p>
Postcondición:	Se ha realizado exitosamente el cobro de los productos y se ha generado la factura para el cliente.
Flujo alternativo	- Si el cliente decide cancelar la compra o modificar algún producto, el cajero realiza las modificaciones necesarias en el sistema y procede con el nuevo total a pagar.

## Parte 2:

### Etapas de Análisis

Modelo de Negocios:

Contexto:

SQ Deportes es una tienda que vende artículos deportivos. El sistema actual de gestión de ventas y stock es manual, lo que genera ineficiencias y errores en la gestión de inventarios, pedidos y ventas. Se propone desarrollar un sistema informático que automatice estas tareas.

Objetivos del Proyecto:

Automatizar el proceso de ventas, desde la consulta de productos hasta la facturación.  
Gestionar el inventario de manera eficiente, manteniendo un registro actualizado del stock.  
Facilitar la generación de reportes de ventas y stock.  
Mejorar la experiencia del cliente proporcionando información precisa y rápida.

Justificación:

La implementación de este sistema reducirá errores humanos, optimizará el tiempo de atención al cliente, mejorará la gestión del inventario y proporcionará información en tiempo real para la toma de decisiones.

Etapas de Diseño

Casos de Uso

#### **CU001 - Cliente consulta al vendedor disponibilidad de un producto (Producto disponible)**

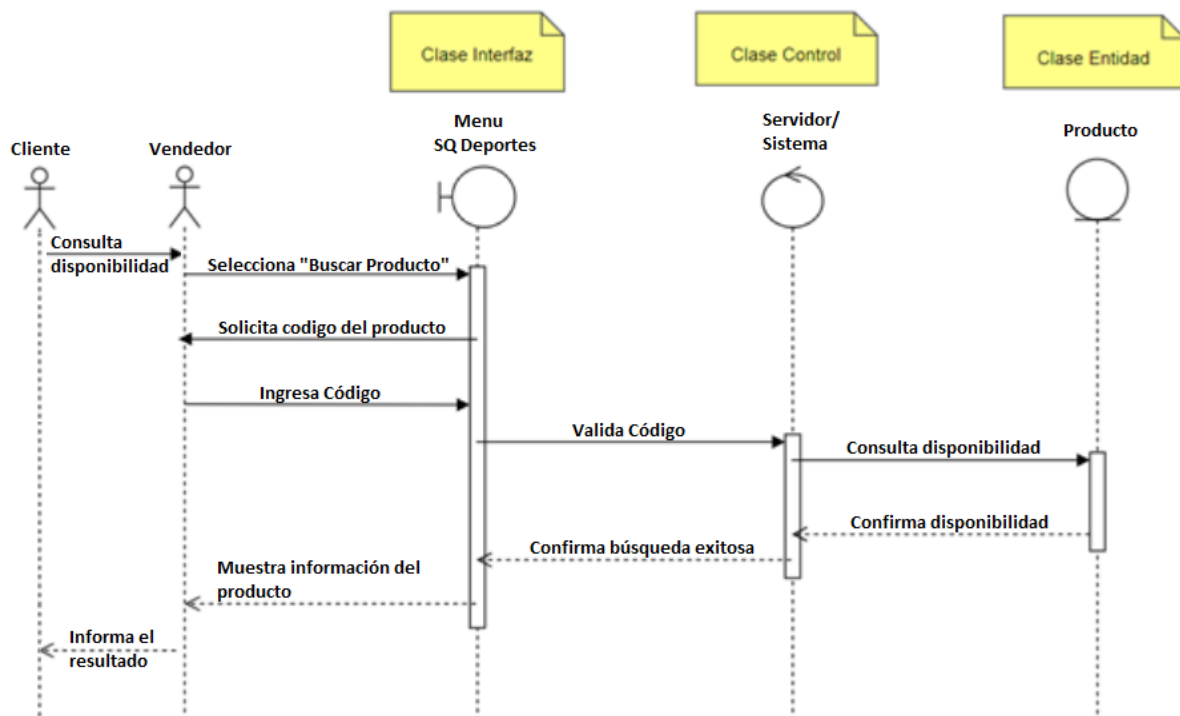
Actores: Cliente, Vendedor

Descripción: Permite al cliente obtener información detallada sobre un producto.

Flujo Principal: El cliente consulta al vendedor disponibilidad de un producto.

- El vendedor selecciona "Buscar producto".
- El sistema pide ingreso del código de producto.
- El vendedor ingresa el código
- El sistema Valida el código
- El sistema consulta disponibilidad de stock a la clase producto.
- La clase Producto confirma disponibilidad.
- El sistema muestra la información del producto.
- El vendedor proporciona la información al cliente.

**Diagrama de secuencia correspondiente al CU001**



Flujo Alternativo:

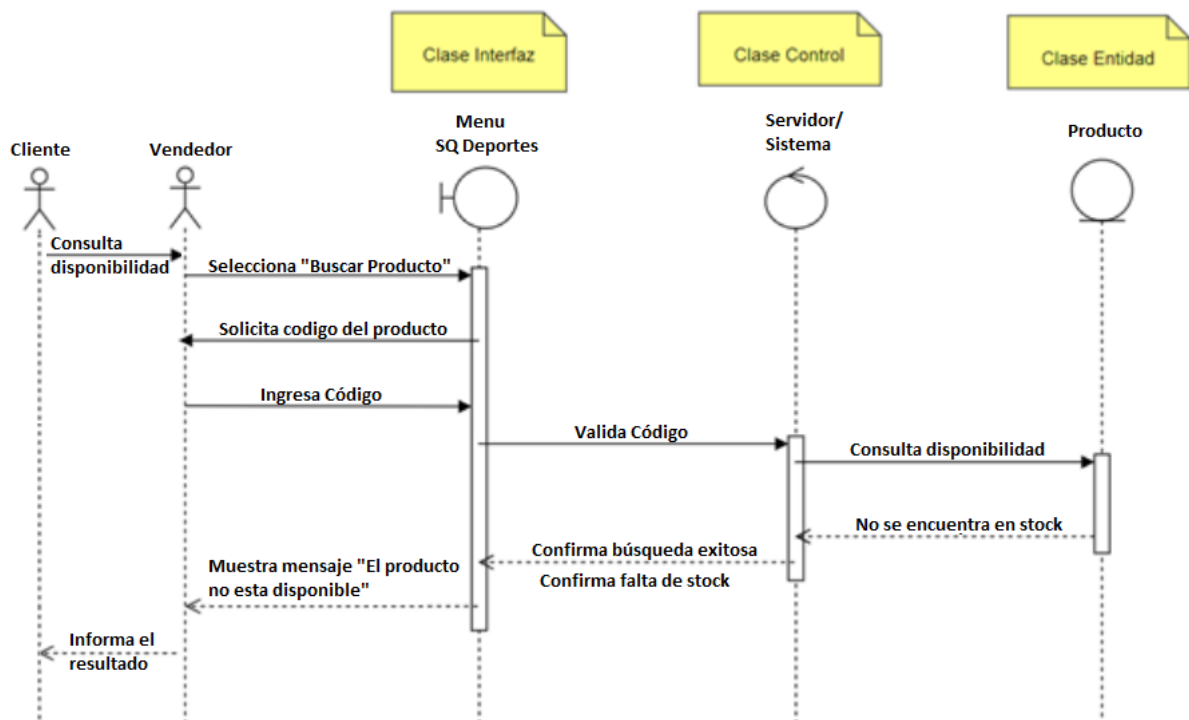
**CU001(Alternativo) - Cliente consulta al vendedor disponibilidad de un producto (Producto NO disponible).**

- El vendedor selecciona "Buscar producto".
- El sistema pide ingreso del código de producto.
- El vendedor ingresa el código
- El sistema Valida el código
- El sistema consulta disponibilidad de stock a la clase producto.
- La clase Producto informa que no hay disponibilidad.
- El sistema notifica al vendedor que el producto no está disponible.
- El vendedor informa al cliente que el producto no está disponible.

Diagrama de secuencia correspondiente al CU001(Alternativo)

En este caso damos por hecho que la búsqueda fue exitosa solo que en la cantidad disponible del producto es 0 lo cual refleje en el diagrama como "Confirma falta de stock" y procede a mostrar un mensaje de "Producto no disponible".





(Los siguientes casos de uso no se diagramaron para no hacer tan extenso el trabajo practico).

#### CU0002 - Vendedor genera pedido de compra

Actores: Vendedor

Descripción: Permite al vendedor generar un pedido de compra.

Flujo Principal:

- El vendedor selecciona "Generar Pedido de Compra".
- El sistema solicita los detalles del pedido.
- El vendedor completa los detalles del pedido.
- El sistema valida la información.
- El sistema Registra el pedido
- El pedido genera una confirmación.
- El sistema Notifica al vendedor que el pedido fue realizado con éxito.

#### CU0003 - Cajero realizar el cobro de los productos

Actores: Cajero

Descripción: Permite al cajero realizar el cobro de los productos.

Flujo Principal:

- El cajero selecciona "Realizar Cobro"
- El sistema solicita datos del cliente
- El cajero ingresa los datos del cliente
- El sistema muestra el pedido de compra.
- El sistema solicita método de pago

- El sistema Procesa el pago.
- El sistema genera la factura.
- El cajero entrega la factura al cliente.

## Etapa de diseño

### Descripción General

El diagrama de clases muestra la estructura estática del sistema, incluyendo las clases principales, sus atributos, métodos y las relaciones entre ellas.

### Clases Principales:

#### 1- Personas(Padre de Cliente, Empleado, Proveedor)

- PK - idPersonas
- nombre
- dirección
- teléfono
- localidad
- provincia

#### 2- Cliente

- PK -idCliente
- DNI
- cuentaCorriente
- Metodos:
- +altaCliente
- +bajaCliente
- +modificarCliente
- +visualizarCliente
- +listarClientes

#### 3- Empleado

- PK -idEmpleado
- CUIL
- rol(cajero, vendedor, administrador)
- Metodos:
- +altaEmpleado
- +bajaEmpleado
- +modificarEmpleado
- +visualizarEmpleado
- +listarEmpleados

#### 4- Proveedor

- PK -idProveedor
- CUIT
- Metodos:
- +altaProveedor

- +bajaProveedor
- +modificarProveedor
- +visualizarProveedor
- +listarProveedores

#### 5- Producto

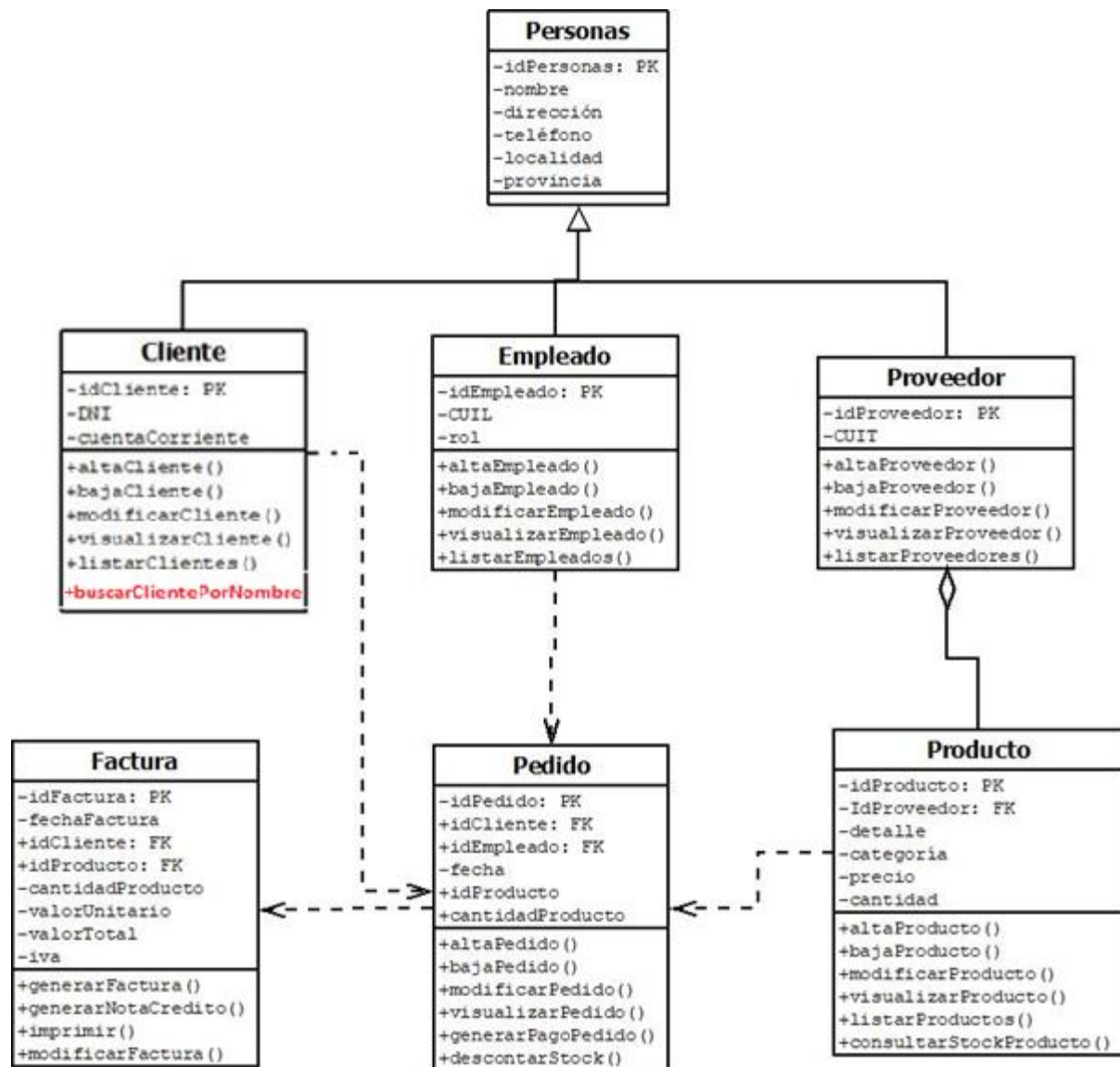
- PK - idProducto
- FK - IdProveedor
- detalle
- categoría
- precio
- cantidad
- Metodos:
- +altaProducto
- +bajaProducto
- +modificarProducto
- +visualizarProducto
- +listarProductos
- +consultarStockProducto

#### 6- Pedido

- PK - idPedido
- FK - idCliente
- FK - idEmpleado
- fecha
- idProducto
- cantidadProducto
- Metodos:
- +altaPedido
- +bajaPedido
- +modificarPedido
- +visualizarPedido
- +generarPagoPedido
- +descontarStock

#### 7- Factura

- idFactura
- fechaFactura
- idCliente
- idProducto
- cantidadProducto
- valorUnitario
- valorTotal
- iva
- Metodos:
- +generarFactura
- +generarNotaCredito
- +imprimir
- +modificarFactura



### Subsistema de Diseño

El subsistema de diseño incluye los componentes y módulos del sistema, destacando cómo se organiza la funcionalidad del sistema en subsistemas lógicos.

### Componentes del Subsistema de Diseño

Gestión de Clientes:

- Módulos: Alta, Baja, Modificación, Visualización, Listado de Clientes.

Gestión de Empleados:

- Módulos: Alta, Baja, Modificación, Visualización, Listado de Empleados.

Gestión de Proveedores:

- Módulos: Alta, Baja, Modificación, Visualización, Listado de Proveedores.

Gestión de Productos:

- Módulos: Alta, Baja, Modificación, Visualización, Listado de Productos, Consulta de Stock.

Gestión de Pedidos:

- Módulos: Alta, Baja, Modificación, Visualización, Generación de Pago, Descuento de Stock.

Gestión de Facturas:

- Módulos: Generación de Factura, Generación de Nota de Crédito, Impresión, Modificación de Factura.

Estos componentes y módulos forman la base del sistema propuesto, proporcionando una estructura clara y organizada para la implementación del sistema de gestión de ventas para SQ Deportes.

## Etapa de Implementación

Requisitos del Sistema

Requerimientos No Funcionales (RNF):

RNF01: El sistema debe estar desarrollado en Java.

RNF02: El sistema debe contar con una base de datos MySQL.

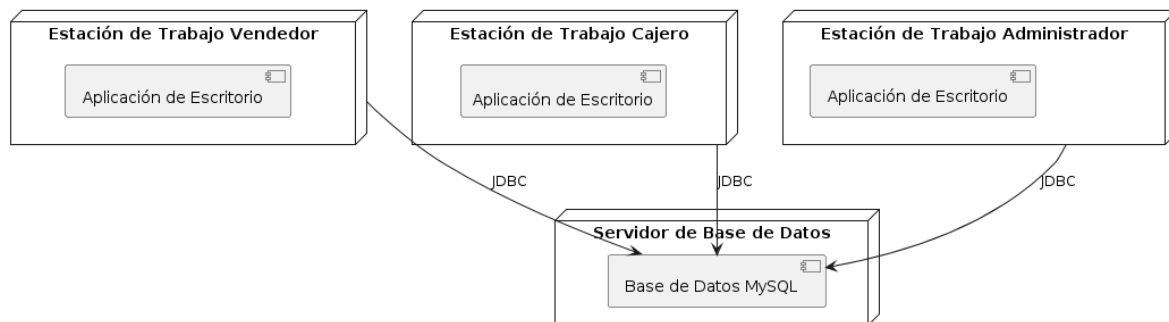
RNF03: El sistema debe utilizar XAMPP como entorno de desarrollo para gestionar la base de datos.

**Servidor de Base de Datos:** Este nodo contiene la base de datos MySQL, gestionada por XAMPP. La base de datos almacena toda la información relacionada con clientes, empleados, proveedores, productos, pedidos y facturas.

**Estación de Trabajo Vendedor:** Los vendedores utilizan una aplicación de escritorio desarrollada en Java para realizar consultas de productos, gestionar pedidos y manejar inventarios. La aplicación se conecta directamente a la base de datos MySQL a través de JDBC (Java Database Connectivity).

**Estación de Trabajo Cajero:** Los cajeros utilizan la misma aplicación de escritorio para realizar cobros, gestionar facturas y procesar pagos. Esta aplicación también se conecta a la base de datos MySQL a través de JDBC.

**Estación de Trabajo Administrador:** Los administradores utilizan la aplicación de escritorio para gestionar la información de clientes, empleados, proveedores y generar reportes, conectándose a la base de datos MySQL mediante JDBC.



## Etapa de Pruebas

En esta sección, se presenta un plan de pruebas detallado adaptado al proyecto de gestión de ventas para SQ Deportes. Se abordarán los casos de uso seleccionados previamente y se definirán las pruebas correspondientes, junto con los criterios de aceptación y el tratamiento de defectos.

### Plan de Pruebas

El plan de pruebas incluirá pruebas unitarias, de integración y del sistema, centrándose en asegurar que el sistema funcione correctamente de acuerdo con los requisitos definidos. A continuación, se detalla el plan de pruebas para los casos de uso CU001, CU002 y CU003.

**Tabla 1: Plan de Pruebas**

Caso de Uso	Código Prueba	Tipo de Prueba	Técnica Propuesta	Observaciones
CU001	CU001C01	Componente	Cobertura	Aplicación de caja blanca al método consultarDisponibilidad() de la clase Producto.
CU001	CU001F02	Componente	Partición de equivalencia-frontera	Aplicación de caja negra al método consultarDisponibilidad() de la clase Producto.
CU001	CU001S03	Sistema	Pruebas funcionales	Verificación de requerimientos funcionales para el caso de uso.
CU002	CU002C01	Componente	Cobertura	Aplicación de caja blanca al método generarPedido() de la clase Pedido.
CU002	CU002F02	Componente	Partición de equivalencia-frontera	Aplicación de caja negra al método generarPedido() de la clase Pedido.
CU002	CU002S03	Sistema	Pruebas funcionales	Verificación de requerimientos funcionales para el caso de uso.
CU003	CU003C01	Componente	Cobertura	Aplicación de caja blanca al método realizarCobro() de la clase Factura.

CU003	CU003F02	Componente	Partición de equivalencia-frontera	Aplicación de caja negra al método realizarCobro() de la clase Factura.
CU003	CU003S03	Sistema	Pruebas funcionales	Verificación de requerimientos funcionales para el caso de uso.

### Casos de Prueba

A continuación, se presenta un caso de prueba específico para el caso de uso CU001, aplicando la técnica de partición de equivalencia y análisis de frontera al método consultarDisponibilidad() de la clase Producto.

**Tabla 2: Requerimientos**

Requerimiento	Descripción
RFS01	El sistema debe mostrar un mensaje de alerta si el producto no está disponible.
RFS02	El sistema debe mostrar un mensaje informativo si el producto está disponible.

### Particiones de Equivalencia Válidas

Clase de Equivalencia Válida	Descripción
CEV1	$0 \leq \text{stock}$
CEV2	$0 \leq \text{stock}$

### Particiones de Equivalencia Inválidas

Clase de Equivalencia Inválida	Descripción
CEI1	$\text{stock} < 0$

### Análisis de Valores de Frontera

**Tabla 3: Análisis de Valores de Frontera**

Clase de Equivalencia	Límite Frontera	Límite Inferior	Límite Superior
CEV1	0	-1	1
CEV2	1	0	2
CEI1	-1	-2	0

### Comportamiento del Sistema

Tabla 4: Comportamiento del Sistema

Dato de Entrada	Comportamiento Esperado	Mensaje Emitido por el Sistema
-2	Error, valor no válido.	Error: stock no puede ser negativo.
-1	Error, valor no válido.	Error: stock no puede ser negativo.
0	Producto no disponible.	Producto no disponible.
1	Producto disponible.	Producto disponible: 1 en stock.
2	Producto disponible.	Producto disponible: 2 en stock.

**Procedimiento de Prueba**

El procedimiento de prueba detalla los pasos a seguir para ejecutar cada caso de prueba. A continuación, se describe el procedimiento para el caso de prueba CU001F02.

Tabla 5: Procedimiento de Prueba para CU001F02

Paso	Descripción
1	Iniciar la aplicación de escritorio y acceder a la funcionalidad de búsqueda de productos.
2	Ingresar el código del producto a consultar.
3	Ejecutar el método consultarDisponibilidad().
4	Verificar el mensaje emitido por el sistema en base a los datos de entrada definidos.
5	Registrar los resultados observados y compararlos con los resultados esperados.

**Tratamiento de Defectos**

Tabla 6: Tratamiento de Defectos

ID Defecto	Descripción del Defecto	Prioridad	Estado	Responsable	Fecha
D001	El sistema permite valores negativos para el stock.	Alta	Abierto	Equipo Dev	2024-06-01
D002	Mensaje incorrecto cuando el stock es 0.	Media	Abierto	Equipo QA	2024-06-01
D003	La aplicación se cierra inesperadamente al consultar stock.	Crítica	Abierto	Equipo Dev	2024-06-01

**Evaluación de Pruebas**

Tabla 7: Evaluación de Pruebas

ID Prueba	Resultado Esperado	Resultado Observado	Estado	Observaciones
CU001F02	Producto no disponible.	Producto no disponible.	Aprobado	Comportamiento conforme a lo esperado.



CU002F02	Pedido generado con éxito.	Producto no disponible.	Aprobado	Comportamiento conforme a lo esperado.
CU003F02	Factura generada y cobro realizado.	Error al generar factura.	Fallido	Error al procesar el pago.

Este plan de pruebas proporciona una guía detallada para verificar y validar la funcionalidad del sistema de gestión de ventas para SQ Deportes. Las pruebas cubren tanto componentes individuales como el sistema completo, asegurando que el software cumpla con los requisitos definidos y funcione correctamente en diversos escenarios.

## Definición de Base de Datos para el sistema

Para el prototipo del sistema de gestión de ventas para SQ Deportes, se utilizará una base de datos relacional MySQL. Esta elección se basa en los requerimientos no funcionales que destacan el buen rendimiento, la flexibilidad y la escalabilidad de MySQL, lo que lo hace adecuado para el desarrollo ordenado del proyecto completo. La base de datos será fundamental para la persistencia de datos clave, como inventario de productos, registros de ventas, información de clientes, proveedores y usuarios.

### Objetivos de la Base de Datos:

Almacenar datos relacionados con productos, ventas, clientes, proveedores y usuarios.  
Mantener un historial completo de las actividades y transacciones realizadas en el sistema.  
Facilitar la generación de informes para analizar el rendimiento de las ventas y el inventario.  
Permitir la identificación de tendencias, problemas y oportunidades para mejorar la eficiencia del negocio.

### Diseño del Prototipo de Base de Datos:

Modelo Vista Controlador (MVC):

Se aplicará el patrón de diseño Modelo Vista Controlador para organizar la estructura del sistema. Las clases asociadas al modelo se traducirán en tablas de la base de datos.

### Representación de Clases en Tablas:

Cada clase del modelo será representada por una tabla en la base de datos.  
Se incluirán tablas para productos, clientes, empleados, pedidos, proveedores y usuarios, entre otras entidades relevantes.  
Relaciones entre Tablas:

Se establecerán relaciones entre las diferentes tablas para reflejar la estructura de datos del sistema. Por ejemplo, una venta estará relacionada con productos y clientes, y un producto estará relacionado con proveedores.

### Tablas del Prototipo de Base de Datos:

**Tabla de Productos:**

Contendrá información sobre los productos disponibles en la tienda, como nombre, descripción, precio y cantidad en stock.

**Tabla de Clientes:**

Almacenará datos de los clientes, incluyendo nombre, dirección, número de teléfono y dirección de correo electrónico.

**Tabla de Empleados:**

Registrará información de los empleados de la tienda, como nombre, cargo, número de empleado y dirección de correo electrónico.

**Tabla de Pedidos:**

Mantendrá un registro de los pedidos realizados por los clientes, incluyendo detalles como fecha, cliente, empleado responsable y estado del pedido.

**Tabla de Facturas:**

Almacenará información sobre las facturas generadas para los pedidos, incluyendo detalles como fecha de emisión, cliente, productos vendidos, total y estado de pago.

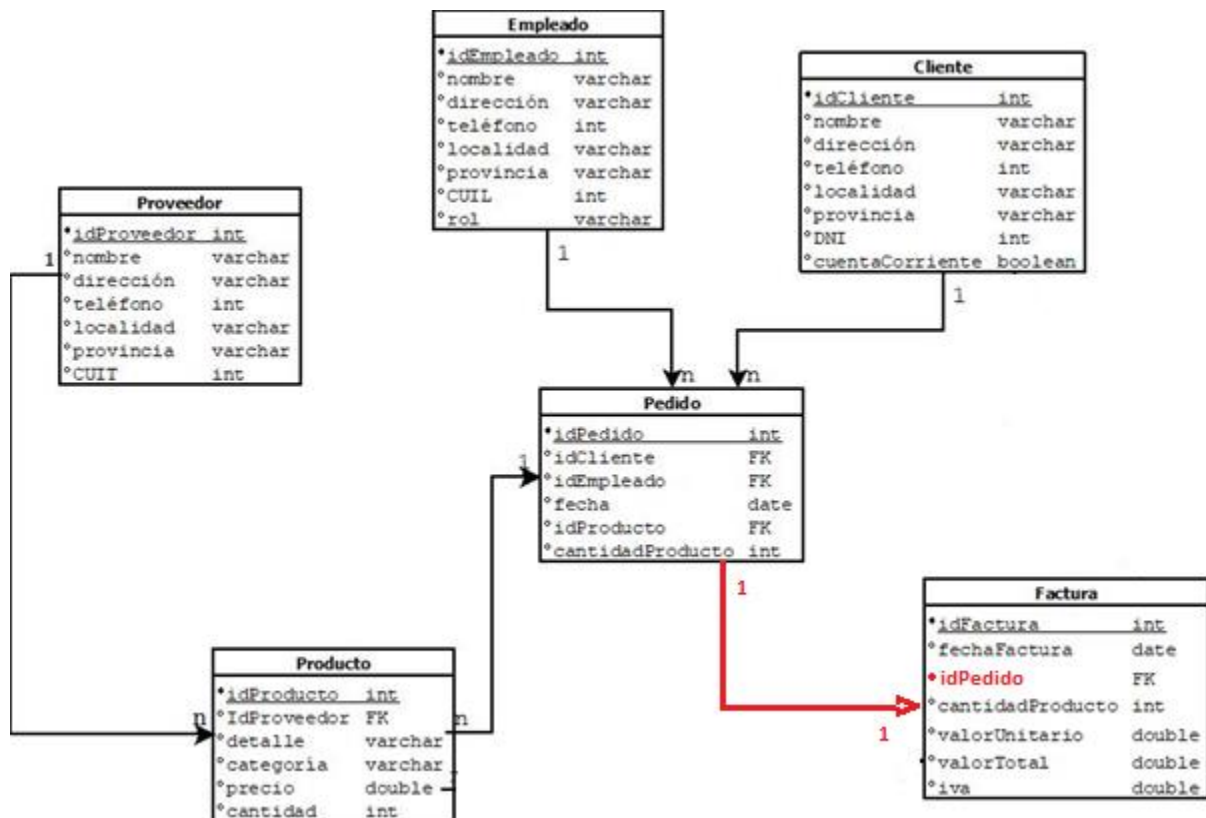
**Tabla de Proveedores:**

Registrará información sobre los proveedores de la tienda, como nombre, dirección, número de contacto y productos suministrados.

**Tabla de Usuarios:**

Mantendrá un registro de los usuarios del sistema, incluyendo nombre de usuario, contraseña y nivel de acceso.

## Diagrama entidad-relación



## Creación de las tablas MySQL.

Modelo de Datos Relacional

```
CREATE DATABASE sqdeportes;
USE sqdeportes;
```

-- Tabla Proveedores

```
CREATE TABLE Proveedores (
    idProveedor INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    direccion VARCHAR(255),
    telefono INT(20),
    localidad VARCHAR(100),
    provincia VARCHAR(100),
    CUIT INT(20)
);
```

-- Tabla Productos

```
CREATE TABLE Productos (
    idProducto INT AUTO_INCREMENT PRIMARY KEY,
    detalle VARCHAR(100) NOT NULL,
```

```
categoria VARCHAR(100),  
precio DECIMAL(10, 2) NOT NULL,  
cantidad INT NOT NULL,  
idProveedor INT,  
FOREIGN KEY (idProveedor) REFERENCES Proveedores(idProveedor)  
);
```

-- Tabla Clientes

```
CREATE TABLE Clientes (  
idCliente INT AUTO_INCREMENT PRIMARY KEY,  
nombre VARCHAR(100) NOT NULL,  
direccion VARCHAR(255),  
telefono INT(20),  
localidad VARCHAR(100),  
provincia VARCHAR(100),  
DNI INT(20),  
cuentaCorriente BOOLEAN  
);
```

-- Tabla Empleados

```
CREATE TABLE Empleados (  
idEmpleado INT AUTO_INCREMENT PRIMARY KEY,  
nombre VARCHAR(100) NOT NULL,  
direccion VARCHAR(255),  
telefono INT(20),  
localidad VARCHAR(100),  
provincia VARCHAR(100),  
rol VARCHAR(50),  
CUIL INT(20)  
);
```

-- Tabla Pedidos

```
CREATE TABLE Pedidos (  
idPedido INT AUTO_INCREMENT PRIMARY KEY,  
fechaPedido DATE NOT NULL,  
cantidad INT(50) NOT NULL,  
idCliente INT,  
idEmpleado INT,  
idProducto INT,  
FOREIGN KEY (idCliente) REFERENCES Clientes(idCliente),  
FOREIGN KEY (idEmpleado) REFERENCES Empleados(idEmpleado),  
FOREIGN KEY (idProducto) REFERENCES Productos(idProducto)  
);
```

-- Tabla Facturas

```
CREATE TABLE Facturas (  
idFactura INT AUTO_INCREMENT PRIMARY KEY,  
fechaFactura DATE NOT NULL,  
idCliente INT,
```

```
idProducto INT,
cantidadProducto INT(50) NOT NULL,
valorunitario DECIMAL(10, 2) NOT NULL,
valortotal DECIMAL(10, 2) NOT NULL,
FOREIGN KEY (idCliente) REFERENCES Clientes(idCliente),
FOREIGN KEY (idProducto) REFERENCES Productos(idProducto)
);
```

Con el código proporcionado, se ejecutó el SQL y creo correctamente las siguientes tablas:

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
<input type="checkbox"/> clientes	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	latin1_swedish_ci	16.0 KB	-
<input type="checkbox"/> empleados	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	latin1_swedish_ci	16.0 KB	-
<input type="checkbox"/> facturas	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	latin1_swedish_ci	32.0 KB	-
<input type="checkbox"/> pedidos	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	latin1_swedish_ci	48.0 KB	-
<input type="checkbox"/> productos	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	latin1_swedish_ci	32.0 KB	-
<input type="checkbox"/> proveedores	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	latin1_swedish_ci	16.0 KB	-
6 tablas	Número de filas	0	InnoDB	latin1_swedish_ci	160.0 KB	0 B

## Inserción, Consulta y Borrado de Registros

### Inserción de Registros

```
INSERT INTO Proveedores (nombre, direccion, telefono, localidad, provincia, CUIT)
VALUES ('Deportes SA', 'Calle Falsa 123', 1234567890, 'Ciudad X', 'Provincia Y', 20304050607),
('Sport Center', 'Avenida Principal 456', 987654321, 'Ciudad W', 'Provincia Z', 30705060809),
('Equipos Deportivos', 'Boulevard Central 789', 1122334455, 'Ciudad V', 'Provincia U', 40906070810);
```

	idProveedor	nombre	direccion	telefono	localidad	provincia	CUIT
<input type="checkbox"/> Editar Copiar Borrar	1	Deportes SA	Calle Falsa 123	1234567890	Ciudad X	Provincia Y	2147483647
<input type="checkbox"/> Editar Copiar Borrar	2	Sport Center	Avenida Principal 456	987654321	Ciudad W	Provincia Z	2147483647
<input type="checkbox"/> Editar Copiar Borrar	3	Equipos Deportivos	Boulevard Central 789	1122334455	Ciudad V	Provincia U	2147483647

```
INSERT INTO Productos (detalle, categoria, precio, cantidad, idProveedor)
VALUES ('Balón de fútbol', 'Deportes', 1500.00, 50, 1),
('Raqueta de tenis', 'Deportes', 2500.00, 30, 2),
('Pesas 5kg', 'Fitness', 300.00, 100, 3);
```

	idProducto	detalle	categoria	precio	cantidad	idProveedor
<input type="checkbox"/> Editar Copiar Borrar	1	Balón de fútbol	Deportes	1500.00	50	1
<input type="checkbox"/> Editar Copiar Borrar	2	Raqueta de tenis	Deportes	2500.00	30	2
<input type="checkbox"/> Editar Copiar Borrar	3	Pesas 5kg	Fitness	300.00	100	3

```
INSERT INTO Clientes (nombre, direccion, telefono, localidad, provincia, DNI, cuentaCorriente)
VALUES ('Juan Pérez', 'Avenida Siempreviva 742', 1234567890, 'Ciudad Z', 'Provincia A',
30123456789, TRUE),
```

('María García', 'Calle 123', 2345678901, 'Ciudad Y', 'Provincia B', 30234567890, FALSE),  
('Pedro López', 'Calle Principal 456', 3456789012, 'Ciudad X', 'Provincia C', 30345678901, TRUE);

←T→	idCliente	nombre	direccion	telefono	localidad	provincia	DNI	cuentaCorriente
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Juan Pérez	Avenida Siempreviva 742	1234567890	Ciudad Z	Provincia A	2147483647	1
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	María García	Calle 123	2147483647	Ciudad Y	Provincia B	2147483647	0
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	Pedro López	Calle Principal 456	2147483647	Ciudad X	Provincia C	2147483647	1

INSERT INTO Empleados (nombre, direccion, telefono, localidad, provincia, rol, CUIL)  
VALUES ('Carlos Gómez', 'Calle 123', 987654321, 'Ciudad Y', 'Provincia B', 'Vendedor', 20123456789),  
('Ana Martínez', 'Avenida Central 456', 876543210, 'Ciudad W', 'Provincia C', 'Cajero', 20234567890),  
('Luis Fernández', 'Boulevard 789', 765432109, 'Ciudad V', 'Provincia D', 'Gerente', 20345678901);

←T→	idEmpleado	nombre	direccion	telefono	localidad	provincia	rol	CUIL
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Carlos Gómez	Calle 123	987654321	Ciudad Y	Provincia B	Vendedor	2147483647
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	Ana Martínez	Avenida Central 456	876543210	Ciudad W	Provincia C	Cajero	2147483647
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	Luis Fernández	Boulevard 789	765432109	Ciudad V	Provincia D	Gerente	2147483647

INSERT INTO Pedidos (fechaPedido, cantidad, idCliente, idEmpleado, idProducto)  
VALUES ('2024-05-19', 10, 1, 1, 1),  
('2024-05-20', 5, 2, 2, 2),  
('2024-05-21', 15, 3, 3, 3);

←T→	idPedido	fechaPedido	cantidad	idCliente	idEmpleado	idProducto
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	2024-05-19	10	1	1	1
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	2024-05-20	5	2	2	2
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	2024-05-21	15	3	3	3

INSERT INTO Facturas (fechaFactura, idPedido, cantidadProducto, valorUnitario, valorTotal)  
VALUES ('2024-05-19', 1, 10, 150.00, 1500.00),  
('2024-05-20', 2, 5, 250.00, 1250.00),  
('2024-05-21', 3, 15, 300.00, 4500.00);

←T→	idFactura	fechaFactura	idPedido	cantidadProducto	valorUnitario	valorTotal
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	2024-05-19	1	10	150.00	1500.00
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	2024-05-20	2	5	250.00	1250.00
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	2024-05-21	3	15	300.00	4500.00

### Consulta de Registros

SELECT \* FROM Proveedores;

←T→	idProveedor	nombre	direccion	telefono	localidad	provincia	CUIT
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Deportes SA	Calle Falsa 123	1234567890	Ciudad X	Provincia Y	2147483647
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	Sport Center	Avenida Principal 456	987654321	Ciudad W	Provincia Z	2147483647
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	Equipos Deportivos	Boulevard Central 789	1122334455	Ciudad V	Provincia U	2147483647

SELECT \* FROM Productos;

←T→	idProducto	detalle	categoria	precio	cantidad	idProveedor
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Balón de fútbol	Deportes	1500.00	50	1
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	Raqueta de tenis	Deportes	2500.00	30	2
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	Pesas 5kg	Fitness	300.00	100	3

SELECT \* FROM Clientes;

←T→	idCliente	nombre	direccion	telefono	localidad	provincia	DNI	cuentaCorriente
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Juan Pérez	Avenida Siempre Viva 742	1234567890	Ciudad Z	Provincia A	2147483647	1
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	María García	Calle 123	2147483647	Ciudad Y	Provincia B	2147483647	0
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	Pedro López	Calle Principal 456	2147483647	Ciudad X	Provincia C	2147483647	1

SELECT \* FROM Empleados;

...

SELECT \* FROM Pedidos;

...

SELECT \* FROM Facturas;

...

### Consultar pedidos de un cliente específico

SELECT \* FROM Pedidos WHERE idCliente = 1;

←T→	idPedido	fechaPedido	cantidad	idCliente	idEmpleado	idProducto
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	2024-05-19	10	1	1	1

### Consultar facturas emitidas en una fecha específica

SELECT \* FROM Facturas WHERE fechaFactura = '2024-05-20';

←T→	idFactura	fechaFactura	idPedido	cantidadProducto	valorunitario	valortotal
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	2024-05-20	2	5	250.00	1250.00

### Consultar productos suministrados por un proveedor específico


SELECT \* FROM Productos WHERE idProveedor = 1;

←T→	idProducto	detalle	categoria	precio	cantidad	idProveedor
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Balón de fútbol	Deportes	1500.00	50	1

## Borrado de Registros

Borrar un proveedor específico

DELETE FROM Proveedores WHERE idProveedor = 1;

				idProveedor	nombre	direccion	telefono	localidad	provincia	CUIT			
<input type="checkbox"/>		Editar		Copiar		Borrar	2	Sport Center	Avenida Principal 456	987654321	Ciudad W	Provincia Z	2147483647
<input type="checkbox"/>		Editar		Copiar		Borrar	3	Equipos Deportivos	Boulevard Central 789	1122334455	Ciudad V	Provincia U	2147483647







Borrar un producto específico

DELETE FROM Productos WHERE idProducto = 1;

←T→				idProducto	detalle	categoria	precio	cantidad	idProveedor
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	Raqueta de tenis	Deportes	2500.00	30	2
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	3	Pesas 5kg	Fitness	300.00	100	3







Borrar un cliente específico

DELETE FROM Clientes WHERE idCliente = 1;

<div>←T→</div>				idCliente	nombre	direccion	telefono	localidad	provincia	DNI	cuentaCorriente
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	María García	Calle 123	2147483647	Ciudad Y	Provincia B	2147483647	0
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	3	Pedro López	Calle Principal 456	2147483647	Ciudad X	Provincia C	2147483647	1

Borrar un empleado específico

DELETE FROM Empleados WHERE idEmpleado = 1;

		idEmpleado	nombre	direccion	telefono	localidad	provincia	rol	CUIL
<input type="checkbox"/>	 Editar  Copiar  Borrar	2	Ana Martínez	Avenida Central 456	876543210	Ciudad W	Provincia C	Cajero	2147483647
<input type="checkbox"/>	 Editar  Copiar  Borrar	3	Luis Fernández	Boulevard 789	765432109	Ciudad V	Provincia D	Gerente	2147483647

Borrar un pedido específico

DELETE FROM Pedidos WHERE idPedido = 1;

...

Borrar una factura específica

...

DELETE FROM Facturas WHERE idFactura = 1;

Código SQL en github mediante el siguiente link:

<https://github.com/rolandoandres22/tp2-seminario-de-practica.git>

## Definiciones de comunicación



## Requerimientos de Comunicación del Sistema

A continuación, se detallan los aspectos clave para asegurar una comunicación eficiente y segura entre los componentes del sistema:

### 1. Entorno de Red

- Topología de Red: Se mantendrá una topología de red estrella, con dispositivos conectados a un switch central.
- Segmentación de Red: Utilización de VLANs para separar el tráfico entre servidores de base de datos y estaciones de trabajo.
- Seguridad de Red: Firewalls y sistemas de detección de intrusos (IDS) para proteger contra accesos no autorizados.

### 2. Infraestructura Física

- Servidores: Servidores en rack de alta disponibilidad con redundancia en hardware y conexiones de red.
- Estaciones de Trabajo: Computadoras de escritorio robustas y bien mantenidas para los usuarios finales.
- Cableado: Cableado de red de categoría 6 (Cat6) o superior.
- Dispositivos de Red: Switches gestionables de alta capacidad con soporte para QoS.

### 3. Protocolos de Comunicación

- Entre Aplicaciones de Escritorio y la Base de Datos:
  - MySQL: Utilizar el protocolo nativo de MySQL para la comunicación directa entre la aplicación de escritorio y la base de datos.
  - ODBC/JDBC: Como alternativa para la conexión a la base de datos desde aplicaciones de terceros si es necesario.
- Con Otros Sistemas Externos:
  - SOAP/REST: Para la integración con sistemas externos si la aplicación necesita comunicarse con otros servicios o APIs.
  - Correo Electrónico: SMTP para envío de correos y IMAP/POP3 para recepción.

### 4. Control de Enlace de Datos

- Ethernet: Utilizar Ethernet (IEEE 802.3) como el protocolo principal para la capa de enlace de datos.
- Wi-Fi: Opcionalmente, para estaciones de trabajo móviles o portátiles.
- Seguridad de Enlace:
  - WPA3: Para redes Wi-Fi, si se utilizan.
  - MAC Address Filtering: Para controlar el acceso a la red.

## Interacciones entre los Diferentes Componentes

### 1. Aplicaciones de Escritorio y Servidor de Base de Datos

- Conexión Directa: Las aplicaciones de escritorio se conectarán directamente al servidor de base de datos MySQL utilizando el protocolo MySQL.
- Autenticación y Autorización: Gestión de usuarios y permisos en MySQL para asegurar que solo usuarios autorizados accedan a los datos.

- Encriptación: Utilizar SSL/TLS para encriptar las conexiones a la base de datos.

## **2. Integración con Sistemas Externos**

- APIs SOAP/REST: Utilizar APIs para integrarse con sistemas externos como sistemas de pago, inventarios, etc.
- Servicios Web: En caso de necesidad de comunicación con servicios web, la aplicación de escritorio puede consumir servicios REST o SOAP.

## **Protocolos y Estándares**

- MySQL: Para la comunicación de base de datos.
- ODBC/JDBC: Como alternativas para la conexión a la base de datos.
- SOAP/REST: Para la integración con sistemas externos.
- SSL/TLS: Para encriptar las conexiones a la base de datos.
- Ethernet/Wi-Fi: Para la comunicación de red.
- SMTP/IMAP/POP3: Para servicios de correo electrónico.
- WPA3: Para la seguridad de redes Wi-Fi.
- IEEE 802.3: Para comunicaciones Ethernet.

## **Detalle de Comunicación**

### **Conexión de la Aplicación de Escritorio con MySQL**

- Configuración del Cliente: Las aplicaciones de escritorio deben tener configurada la dirección IP del servidor MySQL, el puerto de conexión, y las credenciales de acceso.
- Conexión Segura: Configurar MySQL para usar SSL/TLS, asegurando que las conexiones entre las aplicaciones de escritorio y el servidor de base de datos estén encriptadas.

## **Configuración de la Red**

- Segmentación: Usar VLANs para separar el tráfico de las estaciones de trabajo del tráfico del servidor de base de datos y otros servidores.
- QoS: Configurar Quality of Service (QoS) en los switches para priorizar el tráfico crítico de la base de datos.
- Firewall: Configurar reglas de firewall para permitir solo el tráfico necesario entre las estaciones de trabajo y el servidor de base de datos.

## **Conclusión:**

El sistema debe garantizar una comunicación eficiente y segura entre las aplicaciones de escritorio y el servidor de base de datos. La infraestructura de red debe estar bien segmentada y protegida mediante firewalls y VLANs. El uso de SSL/TLS para encriptar las conexiones a la base de datos es crucial para mantener la seguridad de los datos. Además, la integración con sistemas externos puede realizarse mediante APIs SOAP o REST, según sea necesario.

## **Parte 3:**

# Implementación en código Java

## Introducción

Este proyecto implementa un sistema de gestión para una tienda deportiva llamado SQDeportes. El objetivo del sistema es gestionar de manera eficiente los datos relacionados con clientes, empleados, proveedores, productos, pedidos y facturas. El sistema se basa en la Programación Orientada a Objetos (POO) en Java, utilizando principios como encapsulamiento, herencia, polimorfismo y abstracción. En el ejemplo para este trabajo practico se codificaron las clases Personas, Cliente y menuPrincipal.

## Correcta Utilización de Sintaxis, Tipos de Datos y Estructuras de Control

El código del sistema SQDeportes se adhiere estrictamente a las convenciones de sintaxis de Java, asegurando la legibilidad y mantenibilidad del código. Los tipos de datos utilizados son apropiados para representar la información de los clientes, y las estructuras de control (como bucles y condiciones) garantizan una lógica de programación clara y eficiente.

## Características del Sistema codificado y compilando

Gestión de Clientes:

- Alta de clientes.
- Baja de clientes.
- Modificación de clientes.
- Visualización de detalles de un cliente.
- Listado de todos los clientes.
- Búsqueda de clientes por nombre.

## Gestión de Empleados, Proveedores, Productos, Pedidos y Facturas:

Aunque no se detalla en este documento, se asume una implementación similar para estos módulos, siguiendo los mismos principios de POO.

## Principios de POO Implementados

**Encapsulamiento:** Los atributos de las clases son privados y se accede a ellos mediante métodos públicos (getters y setters) protegiendo así los datos internos de modificaciones indebidas.

**Herencia:** La clase Cliente hereda de la clase abstracta Personas, reutilizando y extendiendo la funcionalidad de la superclase.

**Polimorfismo:** Se pueden sobrescribir métodos de la clase padre en las clases hijas para proporcionar comportamientos específicos.

**Abstracción:** La superclase Personas es abstracta y no puede instanciarse directamente, solo a través de sus subclasses.

## Tratamiento y Manejo de Excepciones

El sistema incluye un manejo robusto de excepciones para asegurar que las entradas del usuario sean válidas. Por ejemplo, el menú principal utiliza un bloque try-catch para manejar excepciones InputMismatchException, asegurando que solo se acepten valores numéricos para las opciones del menú. Esto mejora la usabilidad y evita errores en tiempo de ejecución.

**Disponibilidad de un Menú de Selección**

El sistema ofrece un menú de selección en la interfaz de línea de comandos, permitiendo al usuario interactuar con las funcionalidades del sistema de manera intuitiva. El menú maneja entradas inválidas y guía al usuario a través de las distintas opciones disponibles.

**Algoritmos de Ordenación y Búsqueda**

Para enriquecer el código se decidió agregar al menú principal la sección “Buscar clientes por nombre”, utilizando algoritmos básicos de búsqueda lineal.

**Empleo de Estructuras Condicionales y Repetitivas**

El sistema utiliza estructuras condicionales (if-else y switch) y repetitivas (for, while, do-while) para controlar el flujo del programa. Estas estructuras son esenciales para implementar la lógica de negocio, como buscar y ordenar clientes.

**Utilización de Constructores para Inicializar Objetos**

La clase Cliente incluye un constructor que inicializa sus atributos, utilizando también los atributos heredados de la clase Personas. Esto asegura que cada instancia de Cliente esté correctamente configurada desde su creación.

**Código del Proyecto**

A continuación, se presenta el código completo del proyecto, comenzando con la clase principal y siguiendo con la clase Cliente que implementa las funcionalidades del sistema.

**Clase menuPrincipal:**

Opciones del menú principal.

```
*menuPrincipal.java  Personas.java  Cliente.java
1 package SQDeportes;
2 import java.util.InputMismatchException;
3 import java.util.Scanner;
4
5 //Clase con menú de selección, demostrando empleo de estructuras condicionales y repetitivas
6 public class menuPrincipal {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10
11         Scanner scanner = new Scanner(System.in);
12         boolean exit = false;
13
14         while (!exit) { // Estructura repetitiva para el menú
15
16             System.out.println("*****");
17             System.out.println(" Bienvenidos al Sistema de Gestion de SQDeportes:");
18             System.out.println("*****");
19             System.out.println(" Por favor, Seleccione una opcion:");
20             System.out.println("*****");
21             System.out.println(" 1. Gestion de Clientes");
22             System.out.println(" 2. Gestion de Empleados");
23             System.out.println(" 3. Gestion de Proveedores");
24             System.out.println(" 4. Gestion de Productos");
25             System.out.println(" 5. Gestion de Pedidos");
26             System.out.println(" 6. Gestion de Facturas");
27             System.out.println(" 0. Salir");
28             System.out.println("*****");
29

```

Bloques Try/Catch y estructura de control switch que llama a los métodos del submenú.

```
*menuPrincipal.java X Personas.java Cliente.java
29
30     try { //Bloque para captura de excepciones solo se pueden ingresar valores numericos al menu
31         int option = scanner.nextInt();
32
33         switch (option) { // Estructura de control condicional para manejar la selección del usuario
34             case 1:
35                 // Gestión de Clientes
36                 gestionarClientes(scanner);
37                 break;
38             case 2:
39                 // Gestión de Empleados
40                 gestionarEmpleados();
41                 break;
42             case 3:
43                 // Gestión de Proveedores
44                 gestionarProveedores();
45                 break;
46             case 4:
47                 // Gestión de Productos
48                 gestionarProductos();
49                 break;
50             case 5:
51                 // Gestión de Pedidos
52                 gestionarPedidos();
53                 break;
54             case 6:
55                 // Gestión de Facturas
56                 gestionarFacturas();
57                 break;
58             case 0:
59                 exit = true;
60                 break;
61             default:
62                 System.out.println("Opcion no valida. Intente nuevamente.");
63         }
64     } catch (InputMismatchException e) {
65         System.out.println("Por favor, ingrese una opcion numerica.");
66         scanner.next();
67     }
68 }
69
70 }
```

Opciones del submenú "Gestión de Clientes".

```
*menuPrincipal.java X Personas.java Cliente.java
70
71 // Menu para gestionar Clientes
72 public static void gestionarClientes(Scanner scanner) {
73     boolean back = false;
74
75     while (!back) { // Estructura repetitiva para el menú
76
77         System.out.println("*****");
78         System.out.println(" Gestion de Clientes:");
79         System.out.println("*****");
80         System.out.println(" 1. Alta de Cliente");
81         System.out.println(" 2. Baja de Cliente");
82         System.out.println(" 3. Modificacion de Cliente");
83         System.out.println(" 4. Visualizacion de Cliente");
84         System.out.println(" 5. Listado de Clientes");
85         System.out.println(" 6. Buscar Clientes por nombre");
86         System.out.println(" 0. Volver al Menu Principal");
87         System.out.println("*****");
88     }
```

Bloques Try/Catch y estructura de control switch que llama a los métodos de la clase Cliente como altaCliente, bajaCliente, etc. Según la opción elegida por el usuario.

```

*menuPrincipal.java × Personas.java Cliente.java
88
89     try { //Bloque para captura de excepciones solo se pueden ingresar valores numericos al menu
90         int option = scanner.nextInt();
91
92         switch (option) { // Estructura de control condicional para manejar la selección del usuario
93             case 1:
94                 Cliente.altaCliente();
95                 break;
96             case 2:
97                 Cliente.bajaCliente();
98                 break;
99             case 3:
100                 Cliente.modificarCliente();
101                 break;
102             case 4:
103                 Cliente.visualizarCliente();
104                 break;
105             case 5:
106                 Cliente.listarClientes();
107                 break;
108             case 6:
109                 Cliente.buscarClientePorNombre();
110                 break;
111             case 0:
112                 back = true;
113                 break;
114             default:
115                 System.out.println("Opcion no valida. Intente nuevamente.");
116         }
117     } catch (InputMismatchException e) {
118         System.out.println("Por favor, ingrese una opción numérica.");
119         scanner.next();
120     }
121 }
122
123

```

Otras opciones del menú principal aun no desarrolladas.

```

123
124 public static void gestionarEmpleados() {
125     // Código para gestionar empleados
126     System.out.println("Elija la opcion Gestion de Cliente que se encuentra funcional");
127 }
128
129 public static void gestionarProveedores() {
130     // Código para gestionar proveedores
131     System.out.println("Elija la opcion Gestion de Cliente que se encuentra funcional");
132 }
133
134 public static void gestionarProductos() {
135     // Código para gestionar productos
136     System.out.println("Elija la opcion Gestion de Cliente que se encuentra funcional");
137 }
138
139 public static void gestionarPedidos() {
140     // Código para gestionar pedidos
141     System.out.println("Elija la opcion Gestion de Cliente que se encuentra funcional");
142 }
143
144 public static void gestionarFacturas() {
145     // Código para gestionar facturas
146     System.out.println("Elija la opcion Gestion de Cliente que se encuentra funcional");
147 }
148
149 }
150

```

**Superclase Personas:**



Abstracción, encapsulamiento, creación de atributos y constructores.

```

1 package SQDeportes;
2
3 //La clase abstracta Personas implementa la abstracción y encapsulamiento
4 public abstract class Personas {
5
6     //Atributos
7     protected int idPersonas;
8     protected String nombre;
9     protected String direccion;
10    protected String telefono;
11    protected String localidad;
12    protected String provincia;
13
14    // Constructor para inicializar objetos
15    protected Personas(int idPersonas, String nombre, String direccion, String telefono, String localidad,
16        String provincia) {
17        super();
18        this.idPersonas = idPersonas;
19        this.nombre = nombre;
20        this.direccion = direccion;
21        this.telefono = telefono;
22        this.localidad = localidad;
23        this.provincia = provincia;
24    }
25

```

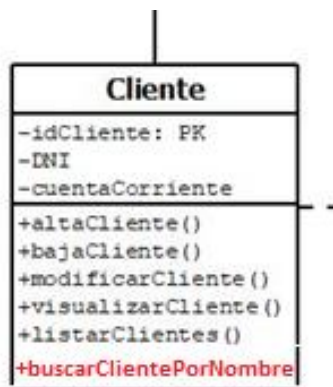
Getter y setter:



```
*menuPrincipal.java  *Personas.java X  Cliente.java
25
26     // Getters y Setters para encapsulamiento
27
28     public int getIdPersonas() {
29         return idPersonas;
30     }
31
32     public void setIdPersonas(int idPersonas) {
33         this.idPersonas = idPersonas;
34     }
35
36     public String getNombre() {
37         return nombre;
38     }
39
40     public void setNombre(String nombre) {
41         this.nombre = nombre;
42     }
43
44     public String getDireccion() {
45         return direccion;
46     }
47
48     public void setDireccion(String direccion) {
49         this.direccion = direccion;
50     }
51
52     public String getTelefono() {
53         return telefono;
54     }
55
56     public void setTelefono(String telefono) {
57         this.telefono = telefono;
58     }
59
60     public String getLocalidad() {
61         return localidad;
62     }
63
64     public void setLocalidad(String localidad) {
65         this.localidad = localidad;
66     }
```

...

**Clase Cliente:**



Subclase Cliente, hereda de Personas. Creación de atributos y constructores.

```

1 package SDeportes;
2 import java.util.ArrayList;
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6
7 //La clase Cliente hereda de Personas, aplicando herencia
8 public class Cliente extends Personas {
9
10     //Atributos
11     private int idCliente;
12     private String DNI;
13     private double cuentaCorriente;
14     private static ArrayList<Cliente> clientes = new ArrayList<>(); //se crea un array para agregar borrar modificar clientes
15
16     // Constructor para inicializar objetos, teniendo en cuenta los atributos de la superclase
17     public Cliente(int idPersonas, String nombre, String direccion, String telefono, String localidad, String provincia,
18         int idCliente, String DNI, double cuentaCorriente) {
19         super(idPersonas, nombre, direccion, telefono, localidad, provincia);
20         this.idCliente = idCliente;
21         DNI = DNI;
22         this.cuentaCorriente = cuentaCorriente;
23     }
24
25 }
  
```

Método altaCliente:

```

28 // Método para dar de alta un cliente con manejo de excepciones
29 public static void altaCliente() {
30     Scanner scanner = new Scanner(System.in);
31     try { //bloque de excepciones
32         System.out.println("*****");
33         System.out.println(" Ingrese el nombre del cliente:");
34         String nombre = scanner.nextLine();
35         System.out.println(" Ingrese el DNI del cliente:");
36         String DNI = scanner.nextLine();
37         System.out.println(" Ingrese la direccion del cliente:");
38         String direccion = scanner.nextLine();
39         System.out.println(" Ingrese el telefono del cliente:");
40         String telefono = scanner.nextLine();
41         System.out.println(" Ingrese la localidad del cliente:");
42         String localidad = scanner.nextLine();
43         System.out.println(" Ingrese la provincia del cliente:");
44         String provincia = scanner.nextLine();
45         System.out.println(" Ingrese la cuenta corriente del cliente:");
46         double cuentaCorriente = scanner.nextDouble();
47
48         Cliente nuevoCliente = new Cliente(clientes.size(), nombre, direccion, telefono, localidad, provincia, clientes.size(), DNI,
49             cuentaCorriente); //agrega el cliente al array
50         System.out.println("*****");
51         System.out.println(" Cliente dado de alta correctamente.");
52         System.out.println("*****");
53     } catch (InputMismatchException e) {
54         System.out.println("*****");
55         System.out.println("Error: Entrada invalida. Por favor, ingrese los datos correctamente.");
56         System.out.println("*****");
57         scanner.nextLine(); // Limpiar el buffer del scanner
58     }
59 }
60
  
```

Método bajCliente:

```

*menuPrincipal.java  *Personas.java  *Cliente.java ×
61 // Método para dar de baja un cliente con manejo de excepciones
62 public static void bajaCliente() {
63     Scanner scanner = new Scanner(System.in);
64     try {
65         System.out.println("*****");
66         System.out.println(" Ingrese el ID del cliente que desea dar de baja:");
67         System.out.println(" Tenga en cuenta que el ID inicia en 0");
68         System.out.println("*****");
69         int idClienteBaja = scanner.nextInt();
70
71         boolean encontrado = false;
72         for (int i = 0; i < clientes.size(); i++) { //estructura repetitiva for
73             Cliente cliente = clientes.get(i);
74             if (cliente.getIdCliente() == idClienteBaja) { //Estructura condicional if
75                 clientes.remove(i); // Eliminando el cliente del array
76                 encontrado = true;
77                 System.out.println("*****");
78                 System.out.println(" Cliente dado de baja correctamente.");
79                 System.out.println("*****");
80                 break;
81             }
82         }
83
84         if (!encontrado) { //estructura condicional caso q no se encuentre
85             System.out.println("*****");
86             System.out.println("Cliente no encontrado.");
87         }
88     } catch (InputMismatchException e) {
89         System.out.println("*****");
90         System.out.println("Error: Entrada invalida. Por favor, ingrese un ID válido.");
91         System.out.println("*****");
92         scanner.nextLine(); // limpiar el buffer del scanner
93     }
94 }
95

```

Método modificarCliente:

```

96 // Método para modificar un cliente con manejo de excepciones
97 public static void modificarCliente() {
98     Scanner scanner = new Scanner(System.in);
99     try {
100         System.out.println("*****");
101         System.out.println(" Ingrese el ID del cliente que desea modificar:");
102         System.out.println(" Tenga en cuenta que el ID inicia en 0");
103         System.out.println("*****");
104         int idClienteModificar = scanner.nextInt();
105
106         boolean encontrado = false;
107         for (Cliente cliente : clientes) {
108             if (cliente.getIdCliente() == idClienteModificar) {
109                 System.out.println("Ingrese el nuevo nombre del cliente:");
110                 scanner.nextLine(); // Consumir el salto de línea pendiente
111                 String nombre = scanner.nextLine();
112                 cliente.setNombre(nombre);
113
114                 System.out.println("Ingrese el nuevo DNI del cliente:");
115                 String dni = scanner.nextLine();
116                 cliente.setDNI(dni);
117
118                 System.out.println("Ingrese la nueva direccion del cliente:");
119                 String direccion = scanner.nextLine();
120                 cliente.setDireccion(direccion);
121
122                 System.out.println("Ingrese el nuevo telefono del cliente:");
123                 String telefono = scanner.nextLine();
124                 cliente.setTelefono(telefono);
125
126                 System.out.println("Ingrese la nueva localidad del cliente:");
127                 String localidad = scanner.nextLine();
128                 cliente.setLocalidad(localidad);
129
130                 System.out.println("Ingrese la nueva provincia del cliente:");
131                 String provincia = scanner.nextLine();
132                 cliente.setProvincia(provincia);
133
134                 System.out.println("Ingrese la nueva cuenta corriente del cliente:");
135                 double cuentaCorriente = scanner.nextDouble();
136                 cliente.setCuentaCorriente(cuentaCorriente);
137
138                 encontrado = true;
139                 System.out.println("*****");
140                 System.out.println(" Cliente modificado correctamente.");
141                 System.out.println("*****");
142                 break;
143             }
144         }
145
146         if (!encontrado) {
147             System.out.println("*****");
148             System.out.println("Cliente no encontrado.");
149         }
150     } catch (InputMismatchException e) {
151         System.out.println("*****");
152         System.out.println("Error: Entrada invalida. Por favor, ingrese los datos correctamente.");
153         System.out.println("*****");
154         scanner.nextLine(); // Limpiar el buffer del scanner
155     }
156 }
157

```

**Método visualizarCliente:**

```

158 // Método para visualizar un cliente con manejo de excepciones
159 public static void visualizarCliente() {
160     Scanner scanner = new Scanner(System.in);
161     try {
162         System.out.println("*****");
163         System.out.println(" Ingrese el ID del cliente que desea visualizar:");
164         System.out.println(" Tenga en cuenta que el ID inicia en 0");
165         System.out.println("*****");
166         int idClienteVisualizar = scanner.nextInt();
167
168         boolean encontrado = false;
169         for (Cliente cliente : clientes) {
170             if (cliente.getIdCliente() == idClienteVisualizar) {
171                 System.out.println("Detalles del cliente:");
172                 System.out.println("ID: " + cliente.getIdCliente());
173                 System.out.println("Nombre: " + cliente.getNombre());
174                 System.out.println("DNI: " + cliente.getDNI());
175                 System.out.println("Direccion: " + cliente.getDireccion());
176                 System.out.println("Telefono: " + cliente.getTelefono());
177                 System.out.println("Localidad: " + cliente.getLocalidad());
178                 System.out.println("Provincia: " + cliente.getProvincia());
179                 System.out.println("Cuenta Corriente: " + cliente.getCuentaCorriente());
180                 encontrado = true;
181                 break;
182             }
183         }
184
185         if (!encontrado) {
186             System.out.println("*****");
187             System.out.println("Cliente no encontrado.");
188         }
189     } catch (InputMismatchException e) {
190         System.out.println("*****");
191         System.out.println("Error: Entrada invalida. Por favor, ingrese un ID válido.");
192         System.out.println("*****");
193         scanner.nextLine(); // Limpiar el buffer del scanner
194     }
195 }
196

```

#### Método listarClientes:

```

197 // Método para listar todos los clientes
198 public static void listarClientes() {
199     System.out.println("*****");
200     System.out.println("Listado de clientes:");
201     System.out.println("*****");
202     if (clientes.isEmpty()) {
203         System.out.println("No hay clientes registrados.");
204     } else {
205         for (Cliente cliente : clientes) {
206             System.out.println("ID: " + cliente.getIdCliente());
207             System.out.println("Nombre: " + cliente.getNombre());
208             System.out.println("DNI: " + cliente.getDNI());
209             System.out.println("Dirección: " + cliente.getDireccion());
210             System.out.println("Teléfono: " + cliente.getTelefono());
211             System.out.println("Localidad: " + cliente.getLocalidad());
212             System.out.println("Provincia: " + cliente.getProvincia());
213             System.out.println("Cuenta Corriente: " + cliente.getCuentaCorriente());
214             System.out.println("-----");
215         }
216     }
217 }
218

```

#### Método buscarClientePorNombre:

```
*menuPrincipal.java  *Personas.java  *Cliente.java X
218
219 // Método para buscar clientes por nombre
220 public static void buscarClientePorNombre() {
221     Scanner scanner = new Scanner(System.in);
222     System.out.println("*****");
223     System.out.println("Ingrese el nombre del cliente que desea buscar:");
224     System.out.println("*****");
225     String nombreBusqueda = scanner.nextLine();
226
227     boolean encontrado = false;
228     for (Cliente cliente : clientes) {
229         if (cliente.getNombre().equalsIgnoreCase(nombreBusqueda)) {
230             System.out.println("Detalles del cliente encontrado:");
231             System.out.println("ID: " + cliente.getIdCliente());
232             System.out.println("Nombre: " + cliente.getNombre());
233             System.out.println("DNI: " + cliente.getDNI());
234             System.out.println("Direccion: " + cliente.getDireccion());
235             System.out.println("Telefono: " + cliente.getTelefono());
236             System.out.println("Localidad: " + cliente.getLocalidad());
237             System.out.println("Provincia: " + cliente.getProvincia());
238             System.out.println("Cuenta Corriente: " + cliente.getCuentaCorriente());
239             encontrado = true;
240             break;
241         }
242     }
243
244     if (!encontrado) {
245         System.out.println("*****");
246         System.out.println("Cliente no encontrado.");
247     }
248 }
249
```

Getter y setter:

```
*menuPrincipal.java  *Personas.java  *Cliente.java X
250 // Getters y Setters para encapsulamiento
251
252 public int getIdCliente() {
253     return idCliente;
254 }
255
256
257 public void setIdCliente(int idCliente) {
258     this.idCliente = idCliente;
259 }
260
261
262 public String getDNI() {
263     return DNI;
264 }
265
266
267 public void setDNI(String dni) {
268     DNI = dni;
269 }
270
271
272 public double getCuentaCorriente() {
273     return cuentaCorriente;
274 }
275
276
277 public void setCuentaCorriente(double cuentaCorriente) {
278     this.cuentaCorriente = cuentaCorriente;
279 }
280
281 }
282
```

Aquí deixo un link de github donde están todos los archivos .java correspondientes al proyecto informático SQ Deportes el código compila correctamente y su menú es perfectamente navegable.

<https://github.com/rolandoandres22/tp3-seminario-de-practica.git>

### Conclusión

El sistema de gestión de clientes para SQDeportes ejemplifica una implementación sólida y completa de los principios de la Programación Orientada a Objetos (POO) en Java. A lo largo del desarrollo del sistema, se ha asegurado una correcta utilización de la sintaxis, tipos de datos y estructuras de control, garantizando así un código legible y eficiente. El manejo de excepciones robustece el sistema, previniendo errores en tiempo de ejecución y mejorando la experiencia del usuario.

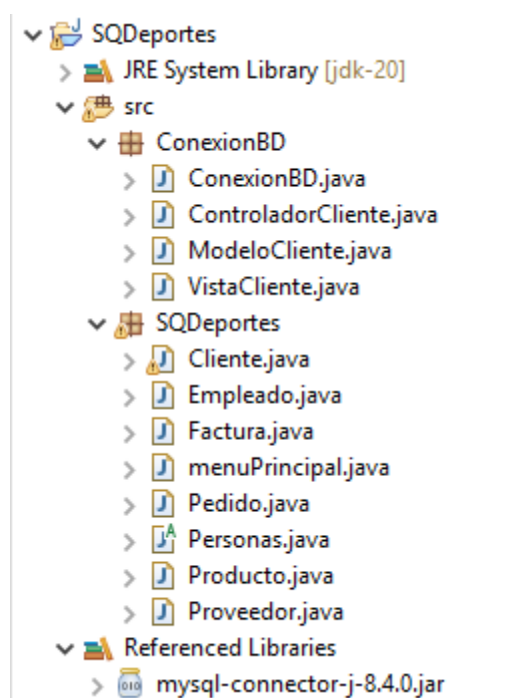
La adecuada aplicación de conceptos clave como encapsulamiento, herencia y abstracción demuestra la capacidad del sistema para ser escalable y mantenible. El encapsulamiento protege los datos internos, mientras que la herencia y el polimorfismo permiten una extensión flexible de las funcionalidades. La abstracción, a través de clases y métodos abstractos, facilita la reutilización y generalización del código.

El sistema ofrece una interfaz de usuario intuitiva mediante un menú de selección, permitiendo la interacción fluida con las diferentes funcionalidades. La inclusión de estructuras condicionales y repetitivas asegura que el flujo del programa se mantenga lógico y coherente. La creación y gestión de objetos se maneja eficientemente, permitiendo operaciones como alta, baja, modificación, visualización y listado de clientes.

Finalmente, la utilización de algoritmos de búsqueda permite una gestión avanzada de los datos de los clientes, mejorando la eficiencia y efectividad del sistema. En resumen, el sistema de gestión de clientes de SQDeportes no solo cumple con los requisitos funcionales, sino que también se adhiere a las mejores prácticas de la programación orientada a objetos, estableciendo una base sólida para futuras extensiones y mejoras.

## Parte 4:

### Conexión a MySQL y MVC



Aquí se aplicaron los conceptos de conexión a bases de datos y patrones MVC (Modelo-Vista-Controlador) al proyecto SQDeportes donde se creó un package llamado ConexionBD y las siguientes clases:

Clase “ConexionBD” para manejar la conexión a la base de datos.

Clase “ModeloCliente” que maneje la lógica de acceso a datos para los clientes.

Clase “VistaCliente” que maneje la interacción con el usuario.

Clase “ControladorCliente” que maneje la lógica de negocio.

Modificación del Menú Principal:



Se modificó la clase menuPrincipal para integrar el controlador de clientes con la base de datos, utilizando los métodos de las clases anterior mencionadas.

## 1- Persistencia y consulta de datos en una base de datos MySQL:

- La clase ConexionBD establece la conexión con la base de datos MySQL usando JDBC.

```
1 package ConexionBD;
2
3 import java.sql.Connection;
4
5
6 public class ConexionBD {
7     // Constantes para la URL de conexión, usuario y contraseña de la base de datos
8     private static final String URL = "jdbc:mysql://localhost:3306/sqdeportes";
9     private static final String USER = "root";
10    private static final String PASSWORD = "";
11
12    // Método estático para establecer una conexión a la base de datos
13    public static Connection conectar() {
14        Connection conexion = null; // Variable para almacenar la conexión
15        try {
16            // Intentar establecer una conexión utilizando los parámetros proporcionados
17            conexion = DriverManager.getConnection(URL, USER, PASSWORD);
18        } catch (SQLException e) {
19            // Capturar y mostrar cualquier excepción de SQL que ocurra durante la conexión
20            System.out.println("Error al conectar a la base de datos: " + e.getMessage());
21        }
22        return conexion; // Devolver la conexión (puede ser null si ocurrió un error)
23    }
24 }
25 }
```

- En ModeloCliente, se implementan métodos para insertar, eliminar, modificar, consultar y listar clientes en la base de datos.

```
12 public class ModeloCliente {
13     private Connection conexion;
14
15     public ModeloCliente(Connection conexion) {
16         this.conexion = conexion;
17     }
18
19     public void insertarCliente(Cliente cliente) throws SQLException {
20         String sql = "INSERT INTO clientes (nombre, DNI, direccion, telefono, localidad, provincia, cuentaCorriente) VALUES (?, ?, ?, ?, ?, ?, ?)";
21         try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
22             stmt.setString(1, cliente.getNombre());
23             stmt.setString(2, cliente.getDNI());
24             stmt.setString(3, cliente.getDireccion());
25             stmt.setString(4, cliente.getTelefono());
26             stmt.setString(5, cliente.getLocalidad());
27             stmt.setString(6, cliente.getProvincia());
28             stmt.setDouble(7, cliente.getCuentaCorriente());
29             stmt.executeUpdate();
30         }
31     }
32
33     public void eliminarCliente(int idCliente) throws SQLException {
34         String sql = "DELETE FROM clientes WHERE idCliente = ?";
35         try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
36             stmt.setInt(1, idCliente);
37             stmt.executeUpdate();
38         }
39     }
40
41     public void modificarCliente(Cliente cliente) throws SQLException {
42         String sql = "UPDATE clientes SET nombre = ?, DNI = ?, direccion = ?, telefono = ?, localidad = ?, provincia = ?, cuentaCorriente = ? WHERE idCliente = ?";
43         try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
44             stmt.setString(1, cliente.getNombre());
45             stmt.setString(2, cliente.getDNI());
46             stmt.setString(3, cliente.getDireccion());
47             stmt.setString(4, cliente.getTelefono());
48             stmt.setString(5, cliente.getLocalidad());
49             stmt.setString(6, cliente.getProvincia());
50             stmt.setDouble(7, cliente.getCuentaCorriente());
51             stmt.setInt(8, cliente.getIdCliente());
52             stmt.executeUpdate();
53         }
54     }
55 }
```

## 2- Correcta aplicación de excepciones para la interacción con la base de datos MySQL:

Se utilizan excepciones (SQLException) en los métodos que interactúan directamente con la base de datos, como insertarCliente, eliminarCliente, modificarCliente, consultarCliente, y listarClientes.

insertarCliente, eliminarCliente y modificarCliente ya se encuentran en la imagen anterior.

```

60 // Método para consultar los datos de un cliente en la base de datos utilizando su ID
61 public Cliente consultarCliente(int idCliente) throws SQLException {
62     String sql = "SELECT * FROM clientes WHERE idCliente = ?";
63     try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
64         stmt.setInt(1, idCliente);
65         ResultSet rs = stmt.executeQuery();
66         if (rs.next()) {
67             // Si se encuentra el cliente, crear y devolver un objeto Cliente con sus datos
68             return new Cliente(
69                 rs.getInt("idCliente"),
70                 rs.getString("nombre"),
71                 rs.getString("direccion"),
72                 rs.getString("telefono"),
73                 rs.getString("localidad"),
74                 rs.getString("provincia"),
75                 rs.getInt("idCliente"),
76                 rs.getString("DNI"),
77                 rs.getDouble("cuentaCorriente")
78             );
79         }
80     }
81     return null; // Si no se encuentra el cliente, devolver null
82 }
83
84 // Método para listar todos los clientes de la base de datos
85 public List<Cliente> listarClientes() throws SQLException {
86     List<Cliente> clientes = new ArrayList<>();
87     String sql = "SELECT * FROM clientes";
88     try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
89         ResultSet rs = stmt.executeQuery();
90         while (rs.next()) {
91             // Agregar cada cliente encontrado a la lista de clientes
92             clientes.add(new Cliente(
93                 rs.getInt("idCliente"),
94                 rs.getString("nombre"),
95                 rs.getString("direccion"),
96                 rs.getString("telefono"),
97                 rs.getString("localidad"),
98                 rs.getString("provincia"),
99                 rs.getInt("idCliente"),
100                 rs.getString("DNI"),

```

### 3- Inclusión pertinente de clases abstractas o interfaces:

La clase abstracta Persona, que es heredada por la clase Cliente, podemos ver que se cumple el punto sobre la inclusión de clases abstractas o interfaces.

### 4- Utilización complementaria de arreglos y de la clase ArrayList:

Se utiliza la clase ArrayList en ModeloCliente para almacenar y devolver listas de clientes en los métodos listarClientes.

Aquí podemos ver que se emplean listas (ArrayList) para manejar colecciones de datos.

```

84 // Método para listar todos los clientes de la base de datos
85 public List<Cliente> listarClientes() throws SQLException {
86     List<Cliente> clientes = new ArrayList<>();
87     String sql = "SELECT * FROM clientes";
88     try (PreparedStatement stmt = conexion.prepareStatement(sql)) {
89         ResultSet rs = stmt.executeQuery();
90         while (rs.next()) {
91             // Agregar cada cliente encontrado a la lista de clientes
92             clientes.add(new Cliente(
93                 rs.getInt("idCliente"),
94                 rs.getString("nombre"),
95                 rs.getString("direccion"),
96                 rs.getString("telefono"),
97                 rs.getString("localidad"),
98                 rs.getString("provincia"),
99                 rs.getInt("idCliente"),
100                 rs.getString("DNI"),
101                 rs.getDouble("cuentaCorriente")
102             ));
103         }
104     }
105     return clientes; // Devolver la lista de clientes
106 }
107 }

```

#### Conclusión.

El código muestra una implementación funcional de persistencia y consulta de datos en una base de datos MySQL, manejo adecuado de excepciones para operaciones de base de datos, y utiliza colecciones como ArrayList para almacenamiento de datos. Aunque no utiliza clases abstractas o interfaces directamente en este código específico, sigue buenas prácticas de diseño modular y separación de responsabilidades entre la vista (VistaCliente), el controlador (ControladorCliente), y el modelo (ModeloCliente).

Link de github donde se subirá un .zip con todo el proyecto de SQDeportes:

<https://github.com/rolandoandres22/tp4-seminario-de-practica.git>