



SISTEMA DE GESTIÓN DE TIERRAS FISCALES DE LA CIUDAD DE NEUQUÉN

Alumno: Alcover, N. Ariel

Legajo: VINF012465

Proyecto para Seminario EFIP 1

Profesor Titular Disciplinar: Virgolini, Pablo Alejandro

Titular Experta: Ferreyra, Ana Carolina

Fecha: 18-MAR-2024 20-JUL-2024

INDICE

[Título Del Proyecto](#)

[Introducción](#)

[Justificación](#)

[Definiciones Del Proyecto Y Del Sistema](#)

[Elicitación](#)

[Conocimiento Del Negocio](#)

[Propuesta De Solución](#)

[Inicio Del Análisis: Casos De Uso](#)

[Etapas de Análisis](#)

[Etapas de Diseño](#)

[Etapas de Implementación](#)

[Etapas de Pruebas](#)

[Interfaz Gráfica](#)

[Definición De Base De Datos Para El Sistema](#)

[Explicación del desarrollo en Java](#)

[Presentación del desarrollo en Java](#)

[Definiciones de Comunicación](#)

[Correcta utilización de sintaxis, tipos de datos, estructuras de control](#)

[Tratamiento y manejo de excepciones](#)

[Adecuada aplicación del encapsulamiento, herencia, polimorfismo y abstracción](#)

[Disponibilidad de un menú de selección](#)

[Empleo de estructuras condicionales y repetitivas](#)

[Declaración y creación de objetos en Java](#)

[Utilización de constructores para inicializar objetos](#)

[Uso de algoritmos de ordenación y búsqueda](#) (opcional según el desarrollo)

4TA. ENTREGA

[Selección del patrón de diseño y justificación](#)

[Explicación del desarrollo, usando un patrón MVC](#)

[Persistencia y consulta de datos en una base de datos MySQL](#)

[Establecer conexiones](#)

[Realizar consultas](#)

[Actualizar registros](#)

[Presentar resultados en la interfaz](#)

[Correcta aplicación de excepciones para la interacción con la base de datos MySQL](#)

[Inclusión pertinente de clases abstractas o interfaces](#)

[Utilización complementaria de arreglos y de la clase *ArrayList*](#)

[Emplear archivos para guardar y recuperar información relevante](#)

[Presentación del proyecto en un video con una duración, aproximadamente, de 3 minutos](#)

[Glosario](#)

TÍTULO DEL PROYECTO

Sistema de gestión de tierras fiscales de la ciudad de Neuquén.

INTRODUCCIÓN

Con el proyecto, se pretende diseñar y desarrollar un sistema de gestión y monitoreo que permita la correcta asignación de los loteos que la municipalidad ofrece, administrar con seguridad los datos personales de beneficiarios y llevar con certeza la contabilidad de los planes de pagos y sus actualizaciones. Representa un paso fundamental hacia la planificación urbana y el cumplimiento de los objetivos de sostenibilidad ambiental para una ciudad que crece exponencialmente.

Actualmente, el personal del organismo carga la totalidad de los datos en planillas de cálculo Excel®. Esto representa un nivel de seguridad nulo en muchos sentidos. Inicialmente, no hay restricción de acceso a los datos personales de los beneficiarios; el valor inicial de los lotes y las cuotas se calculan “a mano” para cada beneficiario siguiendo las reglas de asignación; y actualizando para cada mes los datos por inflación de acuerdo con índices específicos, por lo que el sistema es altamente susceptible a errores humanos. Estos y otros problemas podemos encontrar dado lo precario del sistema de gestión utilizado actualmente.

JUSTIFICACIÓN

La ciudad de Neuquén, como muchas otras ciudades del país, dispone de tierras fiscales sin explotación. La iniciativa de lotear tierras disuade a especuladores y estafadores de apropiarse de terrenos que luego serían vendidos, o entregados para formar asentamientos precarios. La planificación urbana es importante para delinear el tipo de crecimiento que pretende una ciudad; sumado a la posibilidad de que los lotes tengan los servicios básicos, promueve que los ciudadanos puedan acceder a propiedades con las condiciones de higiene y salud básicas que pretendemos de las sociedades del nuevo siglo.

Hemos presenciado, en los últimos años, como con la ausencia de estas políticas, comunidades se han asentado en zonas no aptas para vivir; cómo, por ejemplo, a la vera de un río. Dando como resultado, imágenes catastróficas de pequeñas comunidades perdiendo todo en la corriente, ya que las tierras están asignadas a las crecidas excepcionales de los ríos Neuquén y Limay. Este y otros ejemplos, como los problemas de salud aparejados a la falta de saneamiento urbano hacen que las enfermedades y mortalidad infantil sea alta en las tierras no urbanizadas.

Con el proyecto se espera mejorar la eficiencia del sistema de gestión, disminuir el trabajo duplicado por no tener bases de datos unificadas, en donde cada departamento del organismo tiene que llevar sus propios datos. También se espera mejorar el tiempo de respuesta a consultas de beneficiarios que, muchas veces y, dado el volumen, no es rápido el proceso de cálculo de actualización de cuotas; por no decir completamente ineficiente.

DEFINICIONES DEL PROYECTO Y DEL SISTEMA

El objetivo general del sistema es llevar una correcta asignación de lotes a beneficiarios que cumplan con los requisitos, poder calcular los valores de ingreso, así como las cuotas con certeza. Desarrollar el sistema LOTEQUÉN para que la municipalidad de la ciudad de Neuquén pueda gestionar adecuadamente las tierras fiscales y sus beneficiarios.

En un futuro se espera poder incorporar información sobre el pago de los beneficiarios, pudiendo hacer un seguimiento web. Esto permitirá realizar estadísticas de éxito del plan de urbanización, así como estimar el retorno económico a la sociedad denominado recupero financiero.

ELICITACIÓN

Para la presentación del caso y la formulación de las distintas etapas del desarrollo se llevaron (y llevarán) a cabo entrevistas. Utilizadas para profundizar en las problemáticas y obtener una comprensión más detallada de las necesidades de los usuarios específicos o partes interesadas. Inicialmente di paso a que usuarios de los departamentos de interés me cuenten de que trata su trabajo cotidiano. Mediante preguntas abiertas y explorar en profundidad de temas relevantes, logré interiorizarme de los métodos que utilizaban actualmente para resolver dichos problemas. Queda

para las futuras etapas del trabajo, ir desarrollando las soluciones que un sistema de software puede brindar.

En este punto logré acceder a los datos personales que deben cargarse, reglas de asignación de lotes y valores iniciales, opciones de financiación y requisitos de actualización.

Departamentos “relevantes”

→ administración

→ adjudicaciones

CONOCIMIENTO DEL NEGOCIO

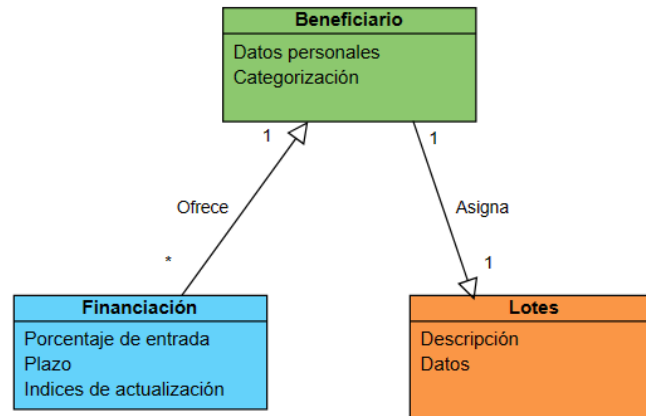
El sistema que se desarrollará permitirá tener un seguimiento de los loteos municipales disponibles, optimizando su asignación y cálculos económicos relevantes para la sostenibilidad del modelo de planificación urbano.

Debe haber una coordinación entre las secretarías, como desarrollo territorial, saneamiento urbano, cooperativas de servicio eléctrico y distribuidoras de gas natural; con el fin de otorgar lotes con servicios básicos. También es importante preestablecer espacios que quedarán en la propiedad pública, para en el futuro, destinarlos a espacios verdes, oficinas públicas, escuelas o salas de salud, entre otros.

Los terrenos ofrecidos son analizados estratégicamente para dar un crecimiento sostenible en los ámbitos de injerencia pública. La denominación debe estar acorde a la nomenclatura de catastro. Los beneficiarios ofrecerán bajo declaración jurada los datos necesarios para el registro, pudiendo realizar una verificación externa de datos clave como informe de dominio catastral.

Por otro lado, el sistema de financiación debe ser acorde a un plan para niveles económicos medios bajos y bajos, por lo que los índices a considerar no tendrán relación a las tasas bancarias. Las actualizaciones seguirán índices de construcción, salarios y valor fiscal de la tierra.

Los procesos involucrados en el desarrollo del software serán la carga de datos de beneficiarios, la asignación de lotes, el cálculo inicial de ingreso al plan de pagos, el cálculo de las cuotas y la actualización de estas.



PROPUESTA DE SOLUCIÓN

A continuación, se presenta una propuesta de solución funcional, la propuesta técnica, la arquitectura planteada para el despliegue y la lista de requerimientos a incluir en el sistema.

Propuesta funcional

Para asegurara la integridad, seguridad y confidencialidad de los datos se establecerán usuarios con accesos diferenciados. Cada clave de usuario podrá modificar los datos que correspondan sin alterar los anteriores. Para favorecer la eficiencia del trabajo administrativo se procederá al cálculo automático de los valores variables de cada período.

Propuesta técnica

El desarrollo del sistema de gestión se realizará en Java, que permite escalar el dominio del negocio. La persistencia se realizará en MySQL, que es una base de datos relacional abierta, flexible y de alto rendimiento para el almacenamiento de datos en tiempo real.

Se pretende ofrecer una base de datos local, con un mínimo de duplicación y backup mensual, sin acceso a internet para evitar la excesiva sofisticación, y los costos extra que genera la protección de estos, dado los tiempos de entrega y presupuesto ajustados.

Requerimientos**R. Funcionales**

Req. Sistema	Descripción
RFS01	Permitir el ingreso mediante usuario y contraseña.
RFS02	Permitir el registro de beneficiarios.
RFS03	Permitir el alta de loteos.
RFS04	Permitir la actualización de los valores de cada lote.
RFS05	Permitir la asignación de lotes.
RFS06	Informar si el beneficiario es apto o no para el loteo social.
RFS07	Informar de las opciones de financiación disponibles para su nivel de ingreso.
RFS08	Establecer financiación según acuerdo con el beneficiario.
RFS09	Calcular cuota de ingreso.
RFS10	Asignar plazo del crédito.
RFS11	Calcular cuotas mensuales.
RFS12	Actualizar mediante índices las cuotas todos los meses.
RFS13	Modificar los datos de beneficiarios.

R. No Funcionales

Req. Sistema	Descripción
RNF01	Estar desarrollado en Java.
RNF02	Contar con base de datos MySQL.
RNF03	Utilizar los servicios de MySQL server.
RNF04	Ofrecer una interfaz amigable para usuarios no familiarizados con la computación.
RNF05	Permitir consultas en la base de datos. Lotes Disponibles, Beneficiarios Rechazados (no aptos), Filtro de Beneficiarios por nivel de ingreso, etc.

R. Candidatos

Req. Sistema	Descripción
RC01	Crear y eliminar usuarios.
RC02	Recuperar contraseña.
RC03	Reasignar lotes.

RC04	Permitir el seguimiento de pagos de los beneficiarios.
RC05	Enviar informes de morosidad.

INICIO DEL ANÁLISIS: CASOS DE USO

Identificación de actores

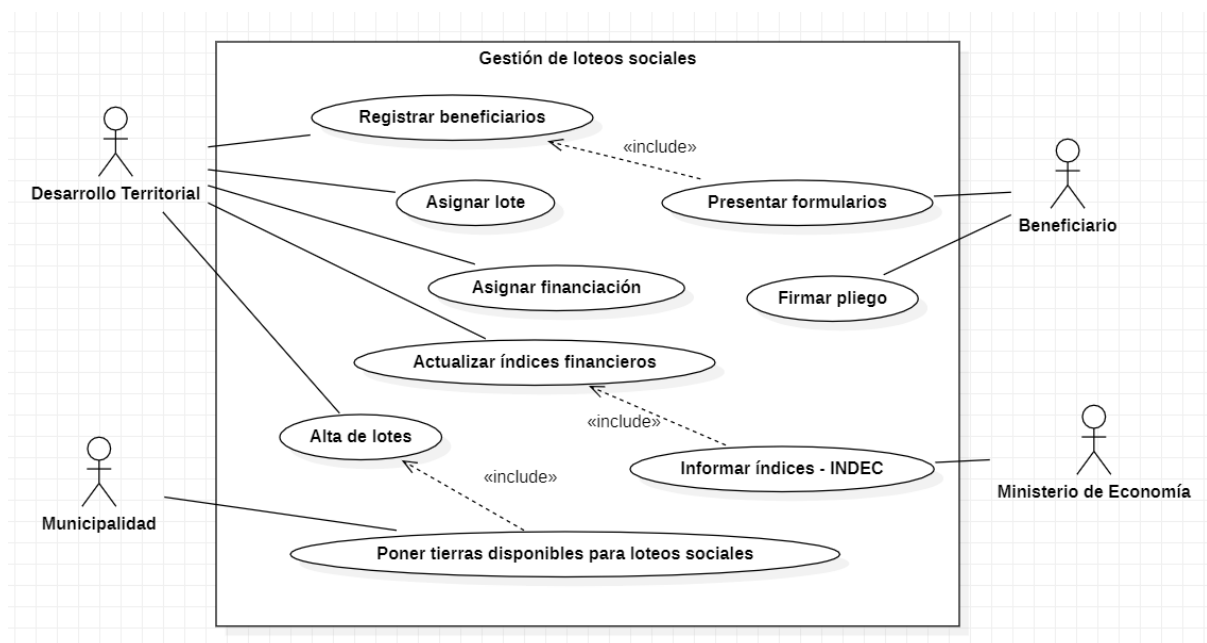
Desarrollo territorial (DLLT): Son los usuarios finales del sistema a desarrollar.

Beneficiario: Son las personas (físicas) a las que se les asignan los lotes.

Municipalidad: Coordina las acciones antes descritas de servicios sobre las tierras fiscales.

Ministerio de economía: El organismo público dedicado a calcular e informar los índices de interés para la economía regional.

Diagrama de casos de uso



Código	Caso de Uso
CU01	Registrar beneficiarios.
CU02	Asignar lote.
CU03	Presentar formularios.
CU04	Asignar financiación.
CU05	Firmar pliego.

CU06	Actualizar índices financieros.
CU07	Informar índices.
CU08	Alta lotes.
CU09	Poner tierras disponibles para loteos sociales.

Trazabilidad

Caso de Uso	Req. Funcional	Actor principal	Paquete del Análisis	Comentario
CU01	RFS02 RFS06 RFS13	DLLT	Beneficiario	Alta posible beneficiario.
CU02	RFS05	DLLT	Lotes	Alta loteo.
CU03	RFS02	Beneficiario	Beneficiario	Datos personales.
CU04	RFS08	DLLT	Lotes	Acuerdo entre el organismo y el beneficiario.
CU05	RFS07 RFS08 RFS09 RFS10	Beneficiario	Beneficiario	Cuota de ingreso.
CU06	RFS11 RFS12	DLLT	Lotes	Las cuotas se actualizan mes a mes.
CU07	RFS12	Ministerio de Economía	Financiación	No perder los valores históricos.
CU08	RFS03	DLLT	Lotes	Trazabilidad entre lotes y loteos.
CU09	RFS03	DLLT	Lotes	Trazabilidad entre loteos y tierra fiscal.

Descripción de Casos de Uso

→ Registrar beneficiario.

Caso de Uso	CU01.
Actores	DLLT.
Referencias	RFS02, RFS06 y RFS13.
Descripción	Permite crear beneficiarios y modificar sus datos.

Precondición	<p>El DLLT debe ingresar con su usuario al sistema.</p> <p>El DLLT debe tener habilitada la carga de beneficiarios.</p> <p>El DLLT debe tener habilitada la modificación de beneficiarios.</p>
Flujo principal	<ol style="list-style-type: none"> 1. El usuario habilitado selecciona la opción "Alta Beneficiario". 2. El sistema muestra la grilla vacía para la carga de datos. 3. El usuario completa los datos que se detallan en el <i>suplemento 1</i>. 4. El usuario confirma la entrada de datos. 5. El sistema valida la información ingresada por el usuario (FP1). 6. El sistema agrega el beneficiario a la base de datos. 7. El sistema muestra un mensaje de confirmación de carga exitosa. 8. El sistema informa si se encuentra apto para el loteo social.
Postcondición	Se agrega al beneficiario a la base de datos, junto con toda la información de formulario detallado en el <i>suplemento 1</i> .
Flujo alternativo	<p>(FP1) Validación de datos falla.</p> <ol style="list-style-type: none"> 1. El sistema muestra un mensaje de error. 2. El sistema muestra los casilleros con errores con sombreado rojo.
Excepciones	<p>Pueden considerarse campos vacíos.</p> <p>Puede no estar definido aún si está apto o no para el loteo social.</p>

→ Asignar lote

Caso de Uso	CU02.
Actores	DLLT.
Referencias	RFS05.
Descripción	Permite asignar lotes (sin asignar) a beneficiarios.
Precondición	<p>El DLLT debe ingresar con su usuario al sistema.</p> <p>El DLLT debe tener habilitada la asignación de lotes.</p> <p>El lote no debe tener asignaciones previas.</p> <p>El beneficiario no debe un lote asignado.</p>
Flujo principal	<ol style="list-style-type: none"> 1. El usuario habilitado selecciona la opción "Asignar Lote". 2. El sistema muestra la grilla vacía para la carga de datos. 3. El usuario completa los datos correspondientes a Loteo y Lote. 4. El usuario confirma la entrada de datos. 5. El sistema valida la información ingresada por el usuario (FP2). 6. El sistema actualiza la base de datos.

Postcondición	Se agrega la asignación a la base de datos, y el lote queda inhabilitado para una nueva asignación.
Flujo alternativo	(FP2) Validación de datos falla. 1. El sistema muestra un mensaje de error. 2. El sistema indica si el beneficiario no está apto para el loteo, o ya tiene un lote asignado, o el lote ya está asignado.
Excepciones	No posee.

→ Presentar formularios

Caso de Uso	CU03.
Actores	Beneficiario.
Referencias	RFS02.
Descripción	Los datos que los usuarios deben cargar en el sistema, para completar la base de datos.
Precondición	El beneficiario fue informado sobre los requisitos para acceder a un loteo social.
Flujo principal	1. El usuario recibe la documentación. 2. El usuario verifica la completitud de esta (FP3). 3. Ambos firman un formulario de inicio de carpeta.
Postcondición	Los datos están disponibles para ingresar a la base de datos del sistema.
Flujo alternativo	(FP3) Faltan informes, datos o formularios. 1. El usuario devuelve la documentación al beneficiario (candidato). 2. No se registran ingresos ni altas.
Excepciones	No posee.

→ Asignar financiación

Caso de Uso	CU04.
Actores	DLLT.
Referencias	RFS08.
Descripción	Entre las opciones que ofrece el loteo social, se acuerda con el beneficiario el plan de pagos.
Precondición	El beneficiario es apto para el loteo social.
Flujo principal	1. Se informa al beneficiario de las opciones disponibles para su caso.

	2. El beneficiario acepta las condiciones. 3. El usuario habilitado selecciona la opción "Asignar Financiación". 4. El sistema muestra la grilla vacía para la carga de datos. 5. El usuario acepta la financiación. 6. El sistema actualiza la base de datos.
Postcondición	El sistema actualiza la base de datos y calcula los pagos que debe hacer el beneficiario para firmar el pliego.
Flujo alternativo	(EX1) Lote vacante 1. El usuario habilitado ingresa al sistema para modificar la asignación del lote.
Excepciones	No hay acuerdo con el plan de financiación, o el beneficiario no posee la cuota inicial solicitada, por lo que no se puede firmar el pliego y el lote queda vacante (EX1).

→ Firmar pliego

Caso de Uso	CU05.
Actores	Beneficiario.
Referencias	RFS07, RFS08, RFS09 y RFS10.
Descripción	Es una carpeta con los datos del beneficiario y el acuerdo de condiciones para acceder al loteo social.
Precondición	El beneficiario acepta las condiciones del crédito, abona la cuota inicial y firma aceptando el lote.
Flujo principal	1. El beneficiario acerca los comprobantes de pago. 2. El usuario completa en el sistema la fecha de pago. 3. El sistema actualiza la base de datos.
Postcondición	El beneficiario queda aprobado, a la espera de su lote.
Flujo alternativo	No posee.
Excepciones	No posee.

→ Actualizar índices financieros

Caso de Uso	CU06.
Actores	DLLT.
Referencias	RFS11 y RFS12.

Descripción	Aunque el loteo social no tiene intereses sobre los créditos, dada la inflación actual, es imposible no actualizar los montos mes a mes para evitar que el organismo se desfinancie.
Precondición	Deben conocerse los índices para cada período y la fecha de referencia ("aprecios de.."). Acceder al boletín oficial.
Flujo principal	<ol style="list-style-type: none"> 1. El usuario habilitado selecciona la opción "Actualizar Índices". 2. El sistema muestra la grilla vacía para la carga de datos. 3. El usuario completa los datos correspondientes al mes y los índices en porcentaje. 4. El usuario confirma la entrada de datos. 5. El sistema actualiza la base de datos. 6. El sistema recalcula las cuotas de ingreso, el costo de los lotes y las cuotas.
Postcondición	Los valores con fecha actual y futura quedan actualizados en la base de datos. Es importante que no actualice las cuotas pagadas con anterioridad de los beneficiarios.
Flujo alternativo	No posee.
Excepciones	No posee.

→ Informar índices

Caso de Uso	CU07
Actores	Ministerio de Economía.
Referencias	RFS12.
Descripción	Son índices públicos, calculados con fines informativos y, utilizados para la tomar decisiones tanto en ámbitos públicos (políticas públicas) como privados (decisión empresarial).
Precondición	<p>Una norma legal establece que índices se utilizarán para la actualización.</p> <ul style="list-style-type: none"> - Índice de la construcción. - Índice casa propia.
Flujo principal	<ol style="list-style-type: none"> 1. Calcular los índices. 2. Informarlos en el boletín oficial.
Postcondición	Actualizar los valores de la economía que toman como referencia estos índices.
Flujo alternativo	No posee.

Excepciones	No posee.
-------------	-----------

→ Alta lotes

Caso de Uso	CU08
Actores	DLLT.
Referencias	RFS03.
Descripción	Se crean nuevos lotes en el sistema para poder asignarlos a beneficiarios.
Precondición	Debe estar habilitado el loteo.
Flujo principal	<ol style="list-style-type: none"> 1. El usuario habilitado selecciona la opción “Alta lotes”. 2. El sistema muestra los loteos por habilitar. 3. El usuario indica cual es el lote que cambiará el estatus a disponible. 4. El usuario confirma la entrada de datos. 5. El sistema actualiza la base de datos.
Postcondición	Aparecen nuevos lotes disponibles, correspondientes al loteo habilitado.
Flujo alternativo	No posee.
Excepciones	No posee.

→ Poner tierras disponibles para loteos sociales

Caso de Uso	CU09
Actores	DLLT.
Referencias	RFS03.
Descripción	Son los terrenos fiscales que la municipalidad dispone para el programa de loteos sociales.
Precondición	Los terrenos ya poseen los servicios.
Flujo principal	<ol style="list-style-type: none"> 1. El usuario habilitado ingresa en la opción “Loteos Sociales”. 2. El sistema muestra una grilla para completar con los datos del loteo. Ubicación, nombre, cantidad de lotes, precio inicial de referencia y estado (habilitado/no habilitado). 3. El usuario confirma la entrada de datos. 4. El sistema valida la información ingresada por el usuario (FP4). 5. El sistema actualiza la base de datos.
Postcondición	Los datos están disponibles para ingresar a la base de datos del sistema. Esperando por la aprobación final para su futura asignación.
Flujo alternativo	(FP4) El sistema muestra un mensaje de error.

	1. El sistema indica sombreando el espacio en blanco si faltara algún dato. → el sistema siempre ingresa loteos como no habilitado, a no ser que el usuario lo marque habilitado.
Excepciones	No aplica.

Suplemento

1. Datos de los beneficiarios

Residencia en la Provincia, RUPROVI, Fecha de inscripción, Trabajo, Ingresos, Situación BCRA, Monto BCRA, Catastro municipal, Catastro provincial, Registro único de la propiedad, Deudor alimentario, Registro violencia género, CCF, Certificado Antecedentes, Hijos

2. Cotitular

Apellido y nombre Cotitular, DNI, CUIL, Fecha de nacimiento, Edad, Sexo, Lugar de nacimiento, Residencia en la Provincia, Trabajo, Ingresos, Situación BCRA, Monto BCRA, Catastro municipal, Catastro provincial, Registro único de la propiedad, Deudor Alimentario, Registro violencia género, CCF, Certificado Antecedentes, Hijos

3. Garante

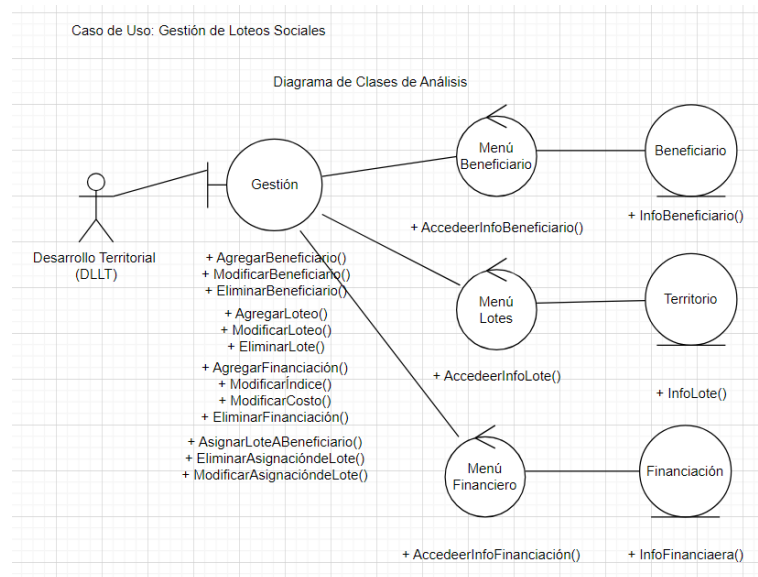
Monto recibo Garante, Discapacidad, Documentación que falta presentar, Observaciones, Ingresos grupo familiar, Posible financiación.

ETAPA DEL ANÁLISIS

Diagrama de Clases del Análisis

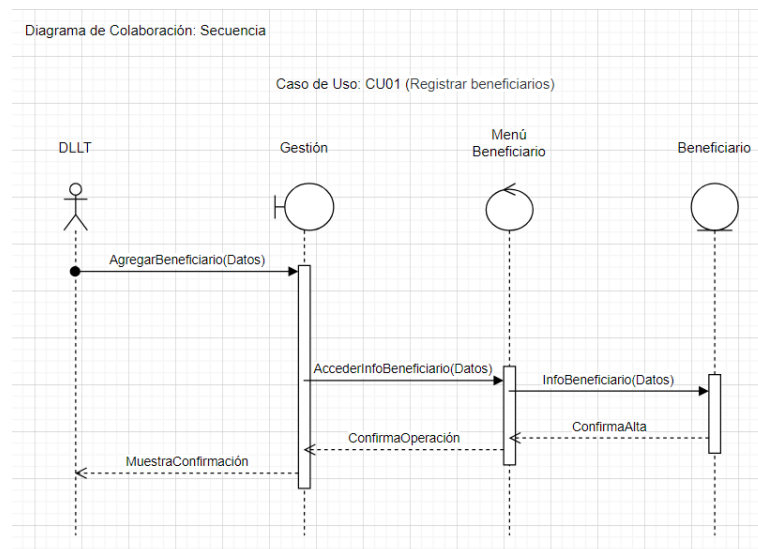
Una clase de análisis representa una abstracción de una o varias clases o subsistemas del diseño de sistema. A través de sus estereotipos, se describe de forma conceptual como el usuario se relaciona con el dominio del problema.

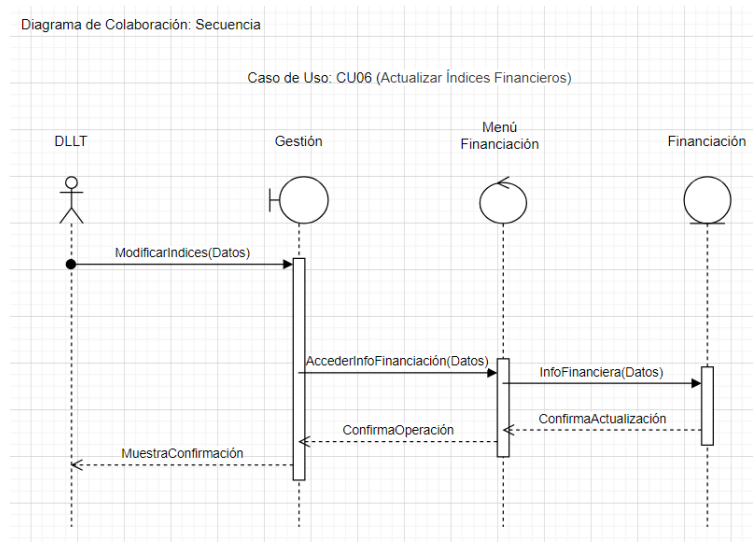
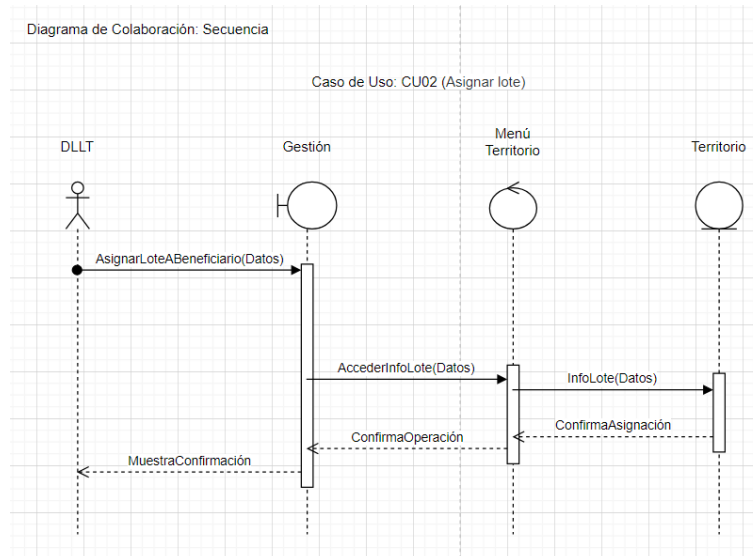
Los usuarios interactúan mediante la interfaz de gestión con las diversas entidades, siguiendo las reglas previstas en el control.



Diagramas de Secuencia

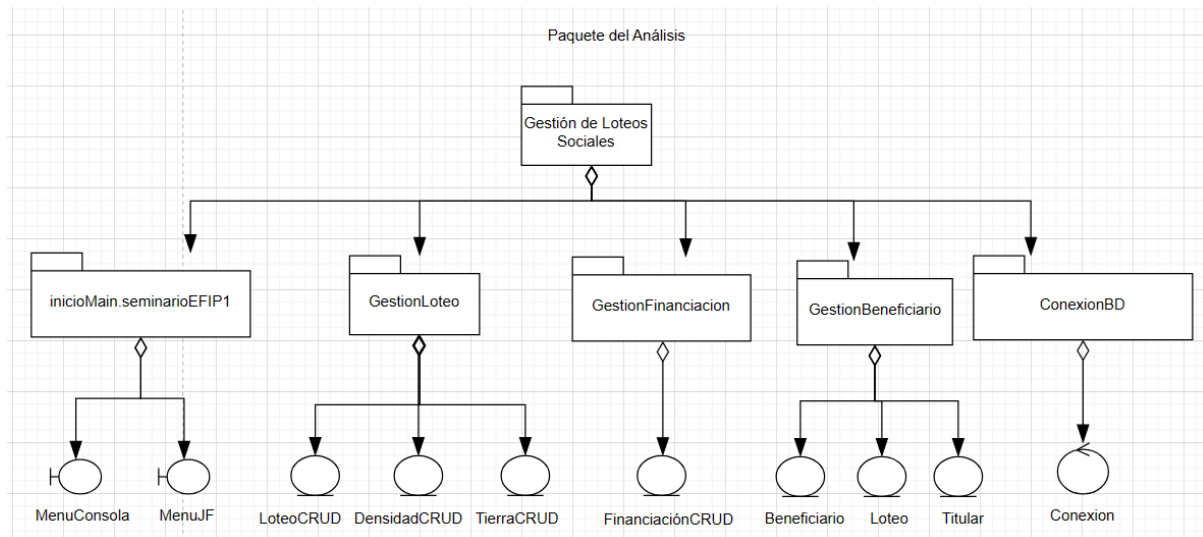
Es uno de los diagramas de colaboración, el cual, permite identificar el intercambio de mensajes entre objetos que representan las operaciones de las clases correspondientes. Dentro de los paréntesis se colocan los valores que requiera dicha operación. El sentido del flujo de los mensajes se representa con una flecha entre los objetos. A continuación, se presenta el diagrama de secuencia para los casos de usos más importantes de cada entidad. El curso alternativo es trivial, dando como resultado un mensaje de error, donde el usuario deberá revisar los datos proporcionados al sistema.





Paquete del Análisis

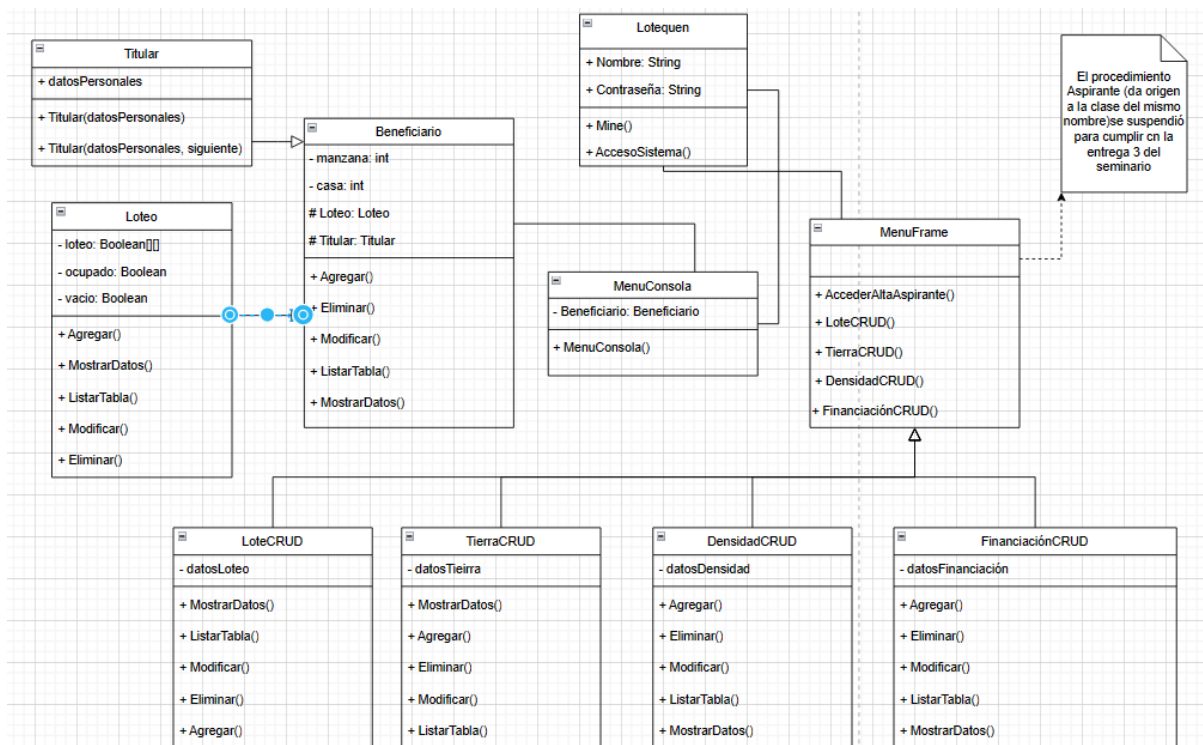
Los paquetes de análisis se utilizan para organizar los artefactos del modelo de análisis en piezas manejables. En nuestro ejercicio contiene los paquetes y sus entidades para describir la forma en que los mismos se relacionan.



ETAPA DE DISEÑO

Diagrama de clases de diseño

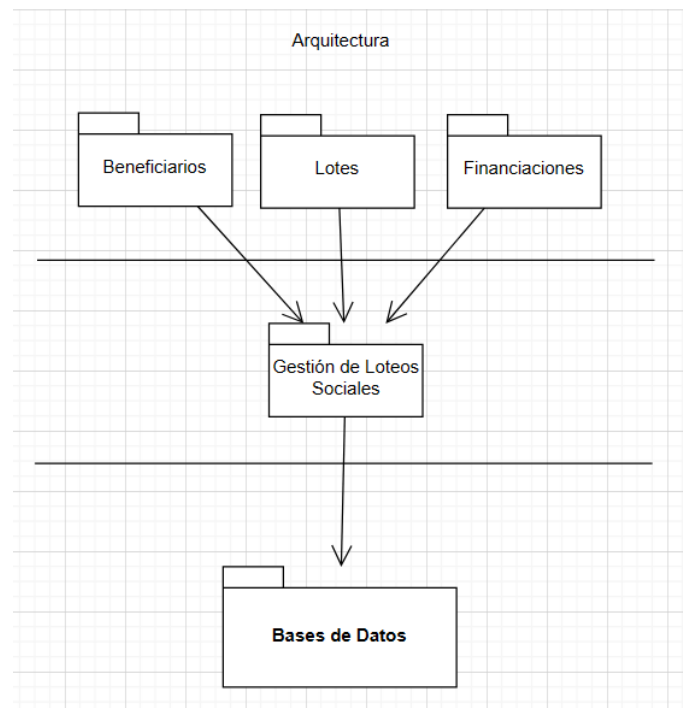
Un diagrama de clases del diseño es una representación visual de las clases, sus atributos y relaciones en un sistema de software.



Nota: Entiendo que no es necesario, o correcto, colocar los estereotipos en las clases del análisis; aunque, yo preferí ponerlo en esta instancia porque me ayuda en la comprensión del diseño.

Subsistema de diseño

Un subsistema tiene que ser cohesivo, aunque débilmente acoplado al resto. Las relaciones entre los paquetes del análisis estarán administradas por el paquete de gestión, la cual, será la capa intermedia entre la base de datos y las demandas de los usuarios.



ETAPA DE IMPLEMENTACIÓN

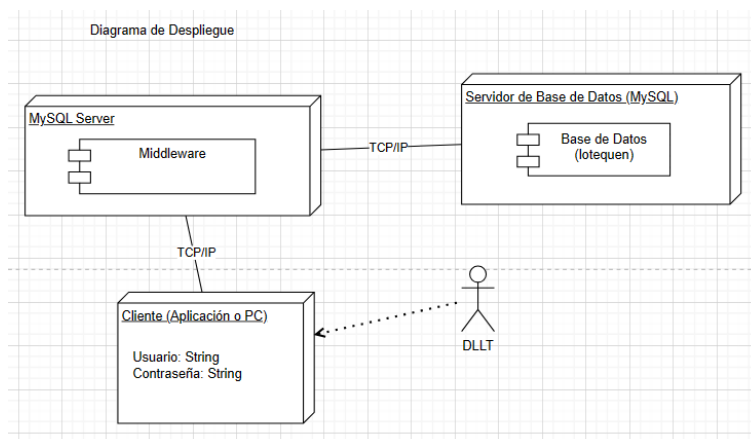
Diagrama de Despliegue

Un diagrama de despliegue representa la distribución de los componentes de un sistema en los nodos físicos (servidores, dispositivos, computadoras) y cómo interactúan entre sí en una implementación real. Es de utilidad para entender cómo se va a desplegar el sistema en un hardware físico y cómo se comunican los componentes.

Para desarrollar el diagrama vamos a describir los siguientes requisitos no funcionales (RNF), los cuales vienen planteados por la solución a desarrollar.

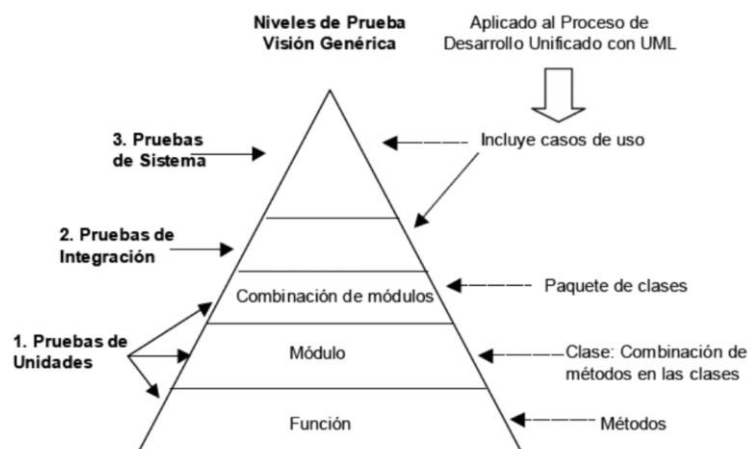
Requerimiento Sistema	Descripción
RNF01	El sistema debe estar desarrollado en Java.
RNF02	El sistema debe contar con una base de datos MySQL.
RNF03	El sistema debe utilizar XAMPP como entorno de desarrollo web de código abierto.

A partir de esta consideración, se presenta el diagrama de despliegue correspondiente.



ETAPA DE PRUEBAS

Todo proceso de prueba de software consiste en tres niveles de prueba: unitarias o de componente, de integración y de sistema. A esto, se le suma la validación que incluye al usuario o cliente, al que llamamos pruebas de aceptación.



Fuente: Braude, 2003.

Plan de pruebas

Si bien el plan de pruebas debe abarcar cada aspecto del sistema, para poder ofrecer un producto de calidad, nos centraremos en el desarrollo de las funcionalidades y los casos de uso más relevantes; intentando mostrar las pruebas más utilizadas en cada ámbito de aplicación.

Referencias de interés

RFS02: Permitir el registro de beneficiarios.

RFS03: Permitir el alta de loteos.

RFS05: Permitir la asignación de lotes.

RFS06: Informar si el beneficiario es apto o no para el loteo social.

RFS08: Establecer financiación según acuerdo con el beneficiario.

CU01: Registrar beneficiario.

CU02: Asignar lote.

CU04: Asignar financiación.

CU08: Alta lotes.

Caso de Uso	RF	Código Prueba	Tipo de Prueba	Técnica Propuesta	Observaciones
CU01	RFS02	CPU01	Unidad	Cobertura	<i>Caja Blanca.</i> Utilizar el método registrarBeneficiario(Datos), de la clase Gestión.
CU02	RFS05	CPU02	Unidad	Estados	<i>Caja negra.</i> Utilizar el método asignarLote(Datos), de la clase Gestión.
CU01	RFS06	CPU03	Unidad	Frontera	<i>Caja negra.</i> Utilizar el método asignarFinanciación(Datos), de la clase Gestión.
CU04	RFS08	CPI01	Integración	Paquetes de Clase	<i>Caja negra.</i> Se ejecuta el caso de uso.
CU08	RFS03	CPS01	Sistema	Desempeño	Mediremos la velocidad a la que los casos de uso particulares se suceden.

UNIDAD O COMPONENTE

→ CPU01

La prueba de caja blanca se realiza sobre el código propiamente dicho. Primero se desglosa el diseño de la aplicación en busca de trayectorias y otras particiones de control y datos. Después se diseñan pruebas que recorran todas o algunas de estas trayectorias y se ejecutan todas las partes o elementos.

→ CPU02

El lote puede transitar por una serie de estados como ser habilitado, libre, asignado, o “no existir” en la base de datos. Ingresando los datos adecuados, se verifica que el estado cambie adecuadamente.

Evento	Estado Inicial	Estado final	Aceptación
Alta loteo	No Existe	Habilitado	El lote figura en la BD como “habilitado”.
Habilitar lote	Habilitado	Libre	El lote figura en la BD como “libre”.
Asignar lote	Libre	Asignado	El lote figura en la BD como “asignado”.
Desasignar lote	Asignado	Libre	El lote figura en la BD como “libre”.

→ CPU03

Las clases de equivalencia válida (cev) estarán dadas por rangos establecidos a partir de los ingresos de los beneficiarios. Los ingresos mayores a 2 millones, o menores a 25 mil pesos corresponden a clase de equivalencia inválidas (cei)

Nota: Los beneficiarios pueden ser cotitulares para alcanzar los ingresos necesarios.

cev	ce válida	Comportamiento esperado	Mensaje emitido
CEV1	$1.000.000 \leq \text{ingreso} < 2.000.000$	Asigna el grupo 1	Beneficiario grupo 1
CEV2	$700.000 \leq \text{ingreso} < 1.000.000$	Asigna el grupo 2	Beneficiario grupo 2
CEV3	$400.000 \leq \text{ingreso} < 700.000$	Asigna el grupo 3	Beneficiario grupo 3
CEV4	$250.000 \leq \text{ingreso} < 400.00$	Asigna el grupo 4	Beneficiario grupo 4

cei	ce inválida	Comportamiento esperado	Mensaje emitido
CEI1	$\text{Ingreso} < 250.000$	Rechaza beneficiario	El beneficiario no alcanza el ingreso mínimo.
CEI2	$2.000.000 < \text{ingreso}$	Rechaza beneficiario	El beneficiario excede el ingreso máximo.

INTEGRACIÓN

→ CPI01

Se simula que un beneficiario consulta al personal de la municipalidad por el financiamiento asignado.

Paquete	Resultados	Retroalimentación
Gestión	Pasa /No pasa	La información es correcta y completa. Motivos de la falla y puntos de mejora.

SISTEMA

→CPS01

Las pruebas del sistema se realizan mientras la aplicación se ejecuta en su entorno requerido.

Se pueden comprobar que las PC con las que cuenta el organismo, son adecuadas para los requisitos del sistema, así como la configurabilidad, ya que muchos organismos trabajan con sistemas operativos bajo la ley de desarrollo libre.

MATRIZ DEL PLAN DE PRUEBAS

Tipo Código	Componente	Integración	Sistema
CPU01	S	N/A	N/A
CPU02	NS	N/A	N/A
CPU03	S	N/A	N/A
CPI01	N/A	S	N/A
CPS01	N/A	N/A	SP

Referencias:

S: Satisfactorio - N: No Satisfactorio - SP: Sin probar - NA: No aplica

TRATAMIENTO DE DEFECTOS

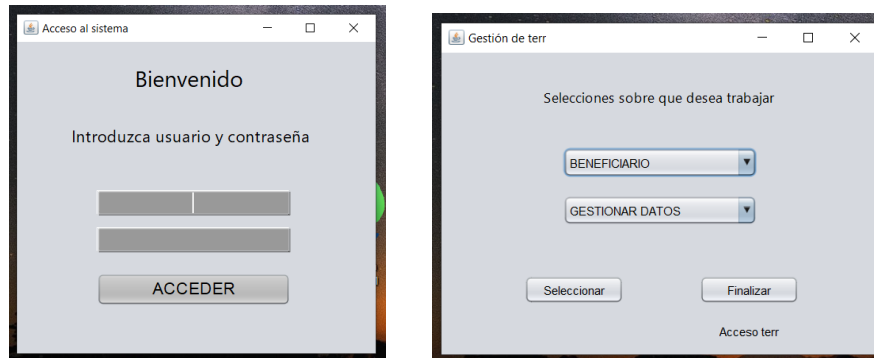
Suponemos que la prueba CPU02 no fue satisfactoria, al ser de componente, no situamos en el desarrollo del código. Se entrega el código al desarrollador para su revisión, junto con el resultado de la prueba.

Informe de Prueba No Satisfactoria.

Código	Evento	Inconformidad	Observaciones	Recibido	Acción Realizada
CPU02	Desasignar lote	No se obtuvo la respuesta esperada	El lote no vuelve a estar "libre"	OK. Nombre	Revisión y modificación realizada.

INTERFAZ GRÁFICA

Su objetivo es experimentar y evaluar la apariencia y la interacción de la interfaz antes de que se desarrolle el producto final. Aquí comparto unas capturas del prototipo que se está elaborando.



DEFINICIÓN DE BASE DE DATOS PARA EL SISTEMA

Recordamos algunos requisitos no funcionales.

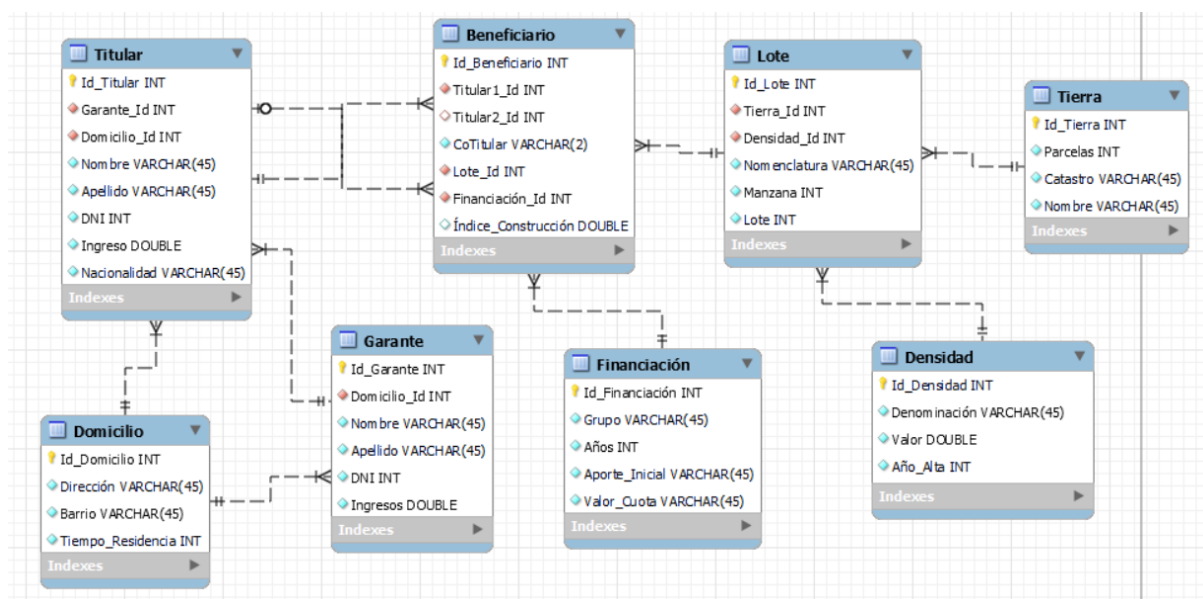
- ✓ Solo se admiten en el programa de loteo social a residentes de la ciudad de Neuquén, con una antigüedad mayor a 5 años.

A continuación, adjuntaré algunas de las capturas a la creación, llenado y vaciado de la base de datos, comenzando por el DER.

El query completo quedará disponible en la nube:

→ <https://github.com/arielna1/Seminario-EFIP1-TP2-.git>

Diagrama de Entidad Relación - DER



Creación de las tablas

```

CREATE TABLE IF NOT EXISTS `lotequen`.`Lote` (
  `Id_Lote` INT NOT NULL AUTO_INCREMENT,
  `Tierra_Id` INT NOT NULL,
  `Densidad_Id` INT NOT NULL,
  `Nomenclatura` VARCHAR(45) NOT NULL,
  `Manzana` INT NOT NULL,
  `Lote` INT NOT NULL,
  PRIMARY KEY (`Id_Lote`),
  CONSTRAINT `Tierra_Id`
    FOREIGN KEY (`Tierra_Id`)
      REFERENCES `lotequen`.`Tierra` (`Id_Tierra`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Densidad_Id`
    FOREIGN KEY (`Densidad_Id`)
      REFERENCES `lotequen`.`Densidad` (`Id_Densidad`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)

CREATE TABLE IF NOT EXISTS `lotequen`.`Tierra` (
  `Id_Tierra` INT NOT NULL AUTO_INCREMENT,
  `Parcelas` INT NOT NULL,
  `Catastro` VARCHAR(45) NOT NULL,
  `Nombre` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_Tierra`))

CREATE TABLE IF NOT EXISTS `lotequen`.`Densidad` (
  `Id_Densidad` INT NOT NULL AUTO_INCREMENT,
  `Denominación` VARCHAR(45) NOT NULL,
  `Valor` DOUBLE NOT NULL,
  `Año_Alta` INT NOT NULL,
  PRIMARY KEY (`Id_Densidad`))

```

Insertión, consulta y borrado de registros-Presentación de las consultas SQL

→ Insert

```

insert into Domicilio (id_Domicilio, dirección, Barrio, Tiempo_Residencia)
values
  (1, 'Suite 65', 'CO', 5),
  (2, 'PO Box 47410', 'ID', 12),
  (3, 'Room 1416', 'VN', 13),
  (4, '17th Floor', 'CN', 20),
  (5, 'PO Box 96367', 'HN', 20),
  (6, 'Suite 12', 'ID', 8),
  (7, 'PO Box 27246', 'VN', 12),
  (8, 'Apt 1045', 'ID', 5),
  (9, 'Apt 689', 'SV', 6),
  (10, 'PO Box 17012', 'CN', 6);

insert into Financiación (Id_Financiación, Grupo, Años, Aporte_Inicial, Valor_Cuota)
values
  (1, 'G1', 7, 6265026.62, 301752.57),
  (2, 'G4', 15, 4701142.49, 271268.38),
  (3, 'G4', 15, 4262362.06, 305009.63),
  (4, 'G2', 10, 3260114.13, 133499.56),
  (5, 'G3', 12, 5307052.58, 160380.09),
  (6, 'G3', 12, 6428244.45, 343410.5),
  (7, 'G2', 10, 3719772.37, 106587.15),
  (8, 'G1', 7, 4234821.83, 335312.56),
  (9, 'G2', 12, 3005584.97, 82674.58),
  (10, 'G2', 12, 6476434.03, 199706.56);

```

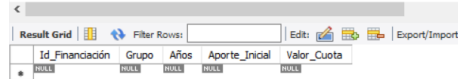
→ Select

```
select * from financiación where Años > 10;
```

	Id_Financiación	Grupo	Años	Aporte_Inicial	Valor_Cuota
▶	2	G4	15	4701142.49	271268.38
	3	G4	15	4262362.06	305009.63
	5	G3	12	5307052.58	160380.09
	6	G3	12	6428244.45	343410.5
	9	G2	12	3005584.97	82674.58
	10	G2	12	6476434.03	199706.56
*	NULL	NULL	NULL	NULL	NULL

→ Delete

```
232  -- -----
233  -- Select
234  -- -----
235
236  • select * from financiación where Años > 10;
237
238  -- -----
239  -- Delete
240  -- -----
241
242  • delete from Financiación;
243
```



The screenshot shows a database management interface. At the top, there is a list of SQL queries with line numbers 232 to 243. The queries are: a comment line, a 'Select' comment line, a 'select * from financiación where Años > 10;' query, another comment line, a 'Delete' comment line, another comment line, and a 'delete from Financiación;' query. Below the queries, there is a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Edit' button, and an 'Export/Import' button. The 'Result Grid' table has five columns: 'Id_Financiación', 'Grupo', 'Años', 'Aporte_Inicial', and 'Valor_Cuota'. The first row of data shows 'Id_Financiación' as '1', 'Grupo' as '1', 'Años' as '1', 'Aporte_Inicial' as '1', and 'Valor_Cuota' as '1'.

DEFINICIONES DE COMUNICACIÓN

Esquemas de Arquitectura de Implementación

Voy a basarme en el estándar ANSI/SPARC y en trabajos prácticos de mí autoría, presentados para Desarrollo de Aplicaciones con Bases de Datos.

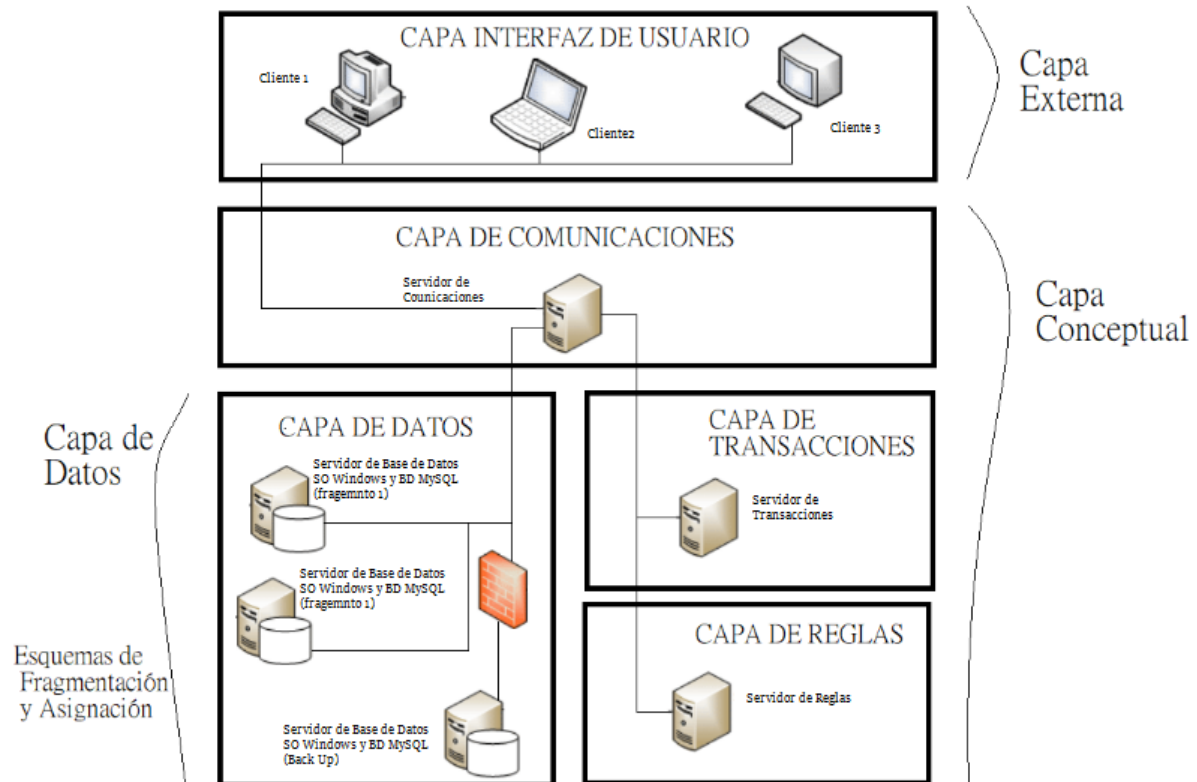
El objetivo principal del estándar es definir un sistema gestor de base de datos con el máximo grado de independencia, separando las aplicaciones de usuarios y las bases de datos física.

El modelo se divide inicialmente en tres capas: Físico, Conceptual y Externo

En el nivel físico se describe como se almacenan los datos (físicamente) en la base de datos y el hardware utilizado.

En el nivel conceptual se describe como se encuentran relacionadas las tablas de la base de datos desarrolladas en los puntos anteriores.

En el nivel externo los usuarios pueden visualizar una parte de la base de datos en particular. Es la única a la cual tienen acceso los usuarios, por lo que aquí se definen los permisos, y se crean las vistas para cada uno de ellos.



Componente	Descripción de la función
Cliente	Son las terminales desde donde los usuarios se conectan a la base de datos, aquí se ubica la Interfaz de Usuario.
Servidor de Comunicaciones	Se ocupa de las comunicaciones entre los equipos cliente y los demás equipos servidores, así como de la comunicación de los servidores entre sí.
Servidor de Transacciones	Es la capa que lleva adelante las interacciones de los usuarios, asegurando que se cumplan las reglas del negocio.
Servidor de Reglas	Gestiona la lógica de las aplicaciones, basándose en las reglas del negocio.
Servidor de Datos	Donde se almacenan y gestionan los datos almacenados.
Servidor Réplica de Datos	Base de datos que funciona como servidor redundante.
Servidor de Back Up	Funciona como tercera redundancia, por si se presentaran fallas.

Los sistemas operativos y la gestión de base de datos están dados por los requisitos no funcionales, definidos en las consignas del presente Trabajo Práctico.

Comunicaciones del Sistema

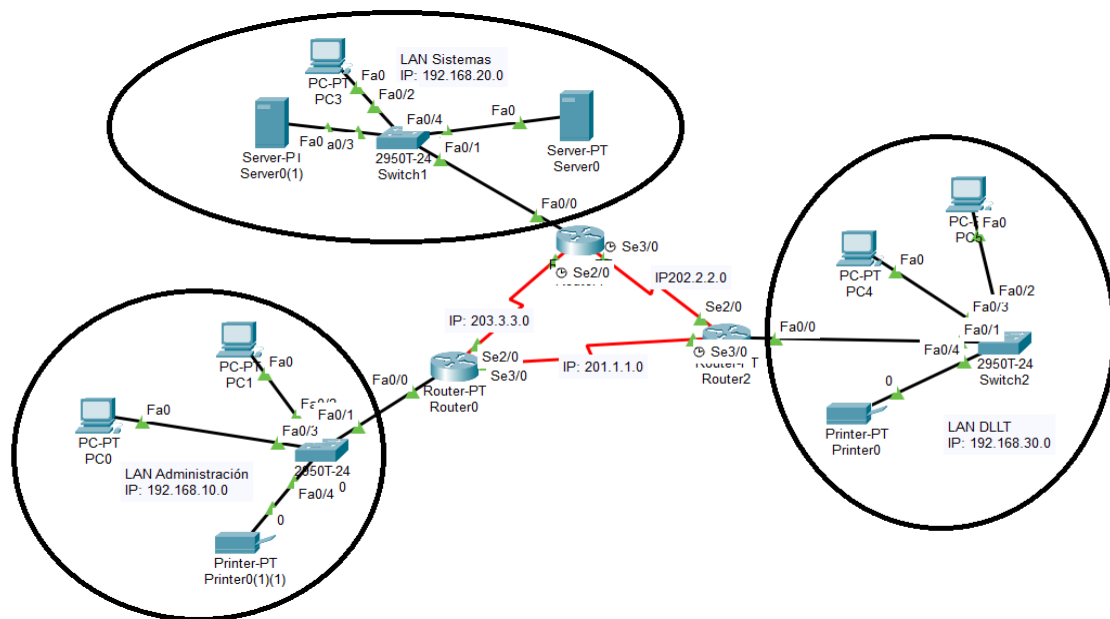
Similar a lo planteado para a arquitectura, en este caso bajo los estándares del modelo OSI, me baso en trabajos de mi autoría, presentados para Comunicaciones.

Los protocolos para las comunicaciones de la red son TCP/IP, ya definidos en el diagrama de despliegue. El direccionamiento consiste en asegurar que cualquier dispositivo conectado a una determinada red cuente con una dirección de IP única. Seguidamente, me apoyaré en el protocolo ICMP que es utilizado para el reporte de errores y consultas de gestión; este protocolo se encuentra justo por encima del protocolo IP en la capa de protocolos TCP/IP.

Para establecer el enlace al medio se aplica el protocolo ARP para redes IPv4 es uno de los protocolos fundamentales de Internet y de las redes locales. A estas direcciones de hardware se las denomina direcciones MAC.

Si bien el protocolo IPv4 está siendo reemplazado paulatinamente por el IPv6, va a funcionar muy bien dado el equipamiento instalado en la sub-secretaría. Paulatinamente, podrían actualizar los routers y demás equipos, si fuera necesaria una mayor velocidad de conexión, o creciera la actividad interna entre departamentos.

Configuré el cableado con cobre para conexiones FastEthernet y Serial DCE para las conexiones Serial, entre Routers.



Tablas

Mascara: 255.255.255.0

Red Administración: IP 192.168.10.0

Dispositivo	Puerto	IP	Getway	Enlace
PC0	FastEthernet0	192.168.10.2	192.168.10.1	Switch0
PC1	FastEthernet0	192.168.10.3	192.168.10.1	Switch0

Printer1(1)	FastEthernet0	192.168.10.4	192.168.10.1	Switch0
Switch0	FastEthernet0/ 1	-	-	PC0
Switch0	FastEthernet0/ 2	-	-	PC1
Switch0	FastEthernet0/ 3	-	-	Router0
Switch0	FastEthernet0/ 4	-	-	Prnter1(1)
Router0	FastEthernet0/ 0	192.168.10.1	-	Switch0
Router0	Serial2/0	202.1.1.1	-	Router1
Router0	Serial3/0	202.2.2.1	-	Router2

Red Sistemas: IP 192.168.20.0

Dispositivo	Puerto	IP	Getway	Enlace
PC3	FastEthernet0	192.168.20.2	192.168.20.1	Switch1
Server0	FastEthernet0	192.168.20.4	192.168.20.1	Switch1
Server0(1)	FastEthernet0	192.168.20.5	192.168.20.1	Switch1
Switch1	FastEthernet0/ 2	-	-	PC3
Switch1	FastEthernet0/ 3	-	-	Server0
Switch1	FastEthernet0/ 4	-	-	Server0(1)
Switch1	FastEthernet0/ 1	-	-	Router1
Router1	FastEthernet0/ 0	192.168.20.1	-	Switch1
Router1	Serial2/0	203.3.3.2	-	Router0
Router1	Serial3/0	201.1.1.2	-	Router2

Red DLLT: IP 192.168.30.0

Dispositivo	Puerto	IP	Getway	Enlace
PC4	FastEthernet0	192.168.30.2	192.168.30.1	Switch2
PC5	FastEthernet0	192.168.30.3	192.168.30.1	Switch2
Printer0	FastEthernet0	192.168.30.4	192.168.30.1	Switch2
Switch2	FastEthernet0/ 1	-	-	Router2
Switch2	FastEthernet0/ 2	-	-	PC4
Switch2	FastEthernet0/ 3	-	-	PC5
Switch2	FastEthernet0/ 4	-	-	Printer0
Router2	FastEthernet0/ 0	192.168.30.1	-	Switch2
Router2	Serial2/0	203.3.3.3	-	Router1
Router2	Serial3/0	202.2.2.2	-	Router0

EXPLICACIÓN DEL DESARROLLO EN JAVA

De acuerdo con los requerimientos no funcionales especificados, el sistema se debe desarrollar en Java. Se debe lograr un código estructurado, legible y modular, para facilitar la comprensión, el mantenimiento y la escalabilidad del proyecto.

Sin perder de vista que el proyecto debe contemplar el desarrollo del sistema completo, en esta instancia se va a trabajar con un prototipo que permita obtener un mínimo producto viable.

Según lo explicado durante la clase, debo centrar el desarrollo del trabajo en explicar las características principales de la POO y, no en el código propiamente dicho; evitando escribir un manual de *mantenimiento o usuario*.

PRESENTACION DEL DESARROLLO EN JAVA

A continuación, nuevamente referencia la ubicación del repositorio:

→ <https://github.com/arielna1/Seminario-EFIP1-TP2-.git>

El prototipo se inicia desde el paquete *inicioMain.seminarioEFIP1*. El mine está ubicado en la clase **Lotequen** para ingresar al sistema es necesario validar el usuario, desde la Base de datos. Dependiendo de que usuario se registra, es a las funciones del prototipo que accede.

```
250
251 CREATE TABLE IF NOT EXISTS `lotequen`.`Usuario` (
252     `Id_Usuario` INT NOT NULL AUTO_INCREMENT,
253     `Nombre` VARCHAR(45) NOT NULL,
254     `Contraseña` VARCHAR(45) NOT NULL,
255     `Estado` VARCHAR(45) NOT NULL,
256     PRIMARY KEY (`Id_Usuario`));
257
258 use lotequen;
259
260 insert into lotequen.usuario (Id_usuario, Nombre, Contraseña, Estado)
261 values
262     (1, 'Ben', '123', 'MenuConsola'),
263     (2, 'Terr', '456', 'MenuJF'),
264     (3, 'Fin', '789', 'Activo'),
265     (4, 'Inac', '159', 'No Activo');
```

Id_Usuario	Nombre	Contraseña	Estado
1	Ben	123	MenuConsola
2	Terr	456	MenuJF
3	Fin	789	Activo
4	Inac	159	No Activo

Para cumplir con las propuestas de la tercera entrega, utilice el acceso del `id_usuario = 1`: Usuario: Ben y Contraseña: 123. Desde aquí se accede a las clases *MenuConsola* que; a su vez, utiliza las clases *Beneficiario*, *Loteo* y *Titular*. A continuación, voy a tomar capturas de los lugares que considero representativos, para ejemplificar algún pilar, característica o funcionalidad solicitada.

Correcta utilización de sintaxis

En este caso ayuda muchísimo trabajar con IDE. Corrige errores de tipeo y advierte conflictos o problemas en la definición de objetos, entre otros; en tiempo real. Solo para casos concretos es necesario compilar la aplicación y recorrer el flujo de programa para encontrar errores de sintaxis..

Por ejemplo, por la falta de definición de un arreglo, es null y al instanciarlo, da error. En ocasiones, la falta de inclusión de estructuras de control de errores hace que el programa se cierre por el error.

Tipos de datos

Hay datos de tipo primitivo como int y boolean, pero también hay estructuras de datos personalizadas, como Titular y Loteo.

```
private int mz, cs;
protected Loteo l;
protected Titular inicio;
private final Scanner entrada = new Scanner(source: System.in);

private final int mz, cs;
private final Boolean[][] loteo;
private final Boolean vacio = true;
private final Boolean ocupado = false;
```

Estructuras de control

Muy utilizado para manejar el flujo dentro de un procedimiento, decidir si hacer o no determinada acción, o cual hacer de las posibles.

```
if (rs.next()) {
    /*cambiando la palabra clave de "Activo" por el Tipo de entidad,
    se puede restringir el acceso de cada usuario. Queda para
    implementaciones futuras.*/
    String estatus = rs.getString(string: "estado");

    if (estatus.equalsIgnoreCase(anotherString: "MenuJF")) {
        dispose();
        new MenuJF().setVisible(b: true);
        JOptionPane.showMessageDialog(parentComponent: null, "Acceso otorgad
        st.close();
    } else if (estatus.equalsIgnoreCase(anotherString: "MenuConsola")) {
        dispose();
        MenuConsola mc = new MenuConsola();
        mc.MenuConsola();
        System.out.println("Acceso otorgado al sistema por Consola para
        st.close();
    } else if (estatus.equalsIgnoreCase(anotherString: "No Activo")) {
        dispose();
        JOptionPane.showMessageDialog(parentComponent: null, "Acceso denegad
        st.close();
    }
    st.close();
} else {
    JOptionPane.showMessageDialog(parentComponent: null, message: "Datos de acc
    txt_user.setText(t: "");
    txt_password.setText(t: "");
}
```



```

switch (seleccion) {
    case 1 -> benA.AltaBen();
    case 2 -> benA.ModificarBen();
    case 3 -> benA.ImprimirLista();
    case 4 -> benA.ML();
    case 5 -> {
        System.out.println(x: "Gracias!, vuelva pronto");
        bandera = 2;
    }
}
}

```

Tratamiento y manejo de excepciones

El tratamiento adecuado de las excepciones favorece al control de flujo y evita que la aplicación falle. Sin la correcta administración de excepciones, al el usuario colocar un índice (número de lote o casa) que no existe en el loteo, el flujo de trabajo se corta y la aplicación falla.

```

try {
    Boolean estado = loteo[mz][cs];
    return estado;
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("No existe el lote seleccionado" + e);
}
}

public void asignarLote(int mz, int cs) {
    if (mz >= 4 || cs >= 4) {
        throw new ArrayIndexOutOfBoundsException(s: "No existe el lote seleccionado");
    } else if (mz < 4 || cs < 4) {
        // ...
    }
}

```

Adecuada aplicación del encapsulamiento, herencia, polimorfismo y abstracción

Bueno, los **encapsulamientos** se pueden ver, por ejemplo en las declaraciones de métodos y atributos.

En las capturas de *Tipos de Datos*, se puede ver el uso de públicos, privados y protegidos.

Por ejemplo, el modificador *protected* en Java ofrece ventajas como: el acceso a los miembros desde la misma clase o clases derivadas; lo que significa que se otorgan ciertos derechos a las subclases; y también proporciona un alto grado de encapsulamiento permitiendo controlar el acceso desde clases externas. Por su parte, un acceso *public* hace que sea visible para todo el paquete, y uno *private* solo lo deja visible para su clase.

En cuanto a la **herencia**, si bien no utilicé claramente esta característica de POO, puedo adelantar una clase del segmento que se encuentra bajo la clase MenuJF. Los objetos creados para dar solución a los CRUD mediante Base de Datos, heredan de las librerías de java.

```

public class DensidadCRUD extends javax.swing.JFrame {

    static String acceso;

    public DensidadCRUD() {
        initComponents();
    }
}

```

El **polimorfismo** es una característica también muy útil, que puede utilizarse de diversas maneras. Aquí, acompaño con una captura del objeto Titular que a veces se comporta según 5 atributos y en otras según 6 atributos. Se puede llamar de igual manera a uno o varios métodos siempre que tengan atributos diferentes; si además están en la misma clase se dice que la clase está sobre cargada. También se podría definir un método como abstracto y sobre escribir el mismo cada vez que sea necesario llamarlo.

```
public Titular(String nombre, String apellido, int dni, int mz,
               int cs, Titular siguiente) {
    this.nombre = nombre;
    this.apellido = apellido;
    this.dni = dni;
    this.mz = mz;
    this.cs = cs;
    this.siguiente = siguiente;
}
public Titular(String nombre, String apellido, int dni, int mz, int cs) {
    this(nombre, apellido, dni, mz, cs, siguiente:null);
}
```

La clase String en Java es un excelente ejemplo de **abstracción**. Representa una secuencia de caracteres (texto) y proporciona métodos para manipularlo, como concatenar, buscar caracteres, etc. Aunque no necesitamos conocer los detalles internos de cómo se almacena y gestiona la cadena, podemos usarla eficazmente en nuestros programas.

```
if (!listaVacia()) {
    String contLista = "Lista actual de beneficiarios \n";
    Titular recorrer = inicio;
    while (recorrer != null) {
        contLista += "nombre: " + recorrer.nombre + " - apellido: "
                    + recorrer.apellido + " - dni: " + recorrer.dni
                    + " - mz: " + recorrer.mz + " - cs: " + recorrer.cs + "\n";
        recorrer = recorrer.siguiente;
    }
}
```

Disponibilidad de un menú de selección

En las clases por consola hay varios, pero voy a referenciar el principal.

```

System.out.println(x: "Por favor selecciones una opción:");
System.out.println(x: "    1. Agregar Beneficiario.");
System.out.println(x: "    2. Modificar Beneficiario.");
System.out.println(x: "    3. Mostrar Lista de beneficiarios.");
System.out.println(x: "    4. Menú de lotes.");
System.out.println(x: "    5. Salir.");
System.out.print(s: "    Selecciones su opción: ");
seleccion = entrada.nextInt();

if (seleccion >= 1 && seleccion <= 5) {
    bandera = 1;
} else {
    System.out.print(s: "Opción no disponible");
    System.out.print(s: "Seleccione una opción disponible");
    System.out.print(s: "    Selecciones su opción: ");
}

```

Empleo de estructuras condicionales y repetitivas

En las clases por consola hay varias, voy a mencionar las principales; ya que incluso están anidadas.

```

do {
    do {
        System.out.println(x: "Por favor selecciones una opción:");
        System.out.println(x: "    1. Agregar Beneficiario.");
        System.out.println(x: "    2. Modificar Beneficiario.");
        System.out.println(x: "    3. Mostrar Lista de beneficiarios.");
        System.out.println(x: "    4. Menú de lotes.");
        System.out.println(x: "    5. Salir.");
        System.out.print(s: "    Selecciones su opción: ");
        seleccion = entrada.nextInt();

        if (seleccion >= 1 && seleccion <= 5) {
            bandera = 1;
        } else {
            System.out.print(s: "Opción no disponible");
            System.out.print(s: "Seleccione una opción disponible");
            System.out.print(s: "    Selecciones su opción: ");
        }
    }

    } while (bandera == 0);

    switch (seleccion) {
        case 1 -> benA.AltaBen();
        case 2 -> benA.ModificarBen();
        case 3 -> benA.ImprimirLista();
        case 4 -> benA.ML();
        case 5 -> {
            System.out.println(x: "Gracias!, vuelva pronto");
            bandera = 2;
        }
    }
} while (bandera != 2);

```

Declaración y creación de objetos en Java

Además de los mencionados en los tipos de datos, también tenemos objetos “abstractos”, importados.

```
private final Beneficiario benA = new Beneficiario();  
private final Scanner entrada = new Scanner(System.in);
```

Utilización de constructores para inicializar objetos

Si no defino un constructor, Java crea uno en tiempo de ejecución. La clase menuConsola no tiene constructor.

Además de la sobre carga de constructores mencionada para Titular, puedo referencia la clase Loteo con su constructor e inicialización.

```
public Loteo(int mz, int cs) {  
    this.loteo = new Boolean[mz][cs];  
    this.mz = mz;  
    this.cs = cs;  
}
```

Uso de algoritmos de ordenación y búsqueda (opcional según el desarrollo).

Al crear mi propio tipo de dato, el objeto Titular, también cree una lista enlazada. Más adelante la referenciaré para cumplir con la entrega 4; al igual que el array Booleano.

Si bien, no implementé los algoritmos que fuimos aprendiendo en los talleres de algoritmos, realicé una pequeña búsqueda en la lista para poder modificar datos ya ingresados a la misma.

```
if (!listaVacia()) {  
    Titular recorrer = inicio;  
    Titular auxiliar = new Titular(nombre, apellido, dni, mz, cs);  
    while (recorrer != null) {  
        if (auxiliar.dni == recorrer.dni) {  
            1.LiberarLote(mz: recorrer.mz, cs: recorrer.cs);  
            recorrer.nombre = auxiliar.nombre;  
            recorrer.apellido = auxiliar.apellido;  
            recorrer.mz = auxiliar.mz;  
            recorrer.cs = auxiliar.cs;  
            1.AsignarLote(mz: auxiliar.mz, cs: auxiliar.cs);  
            System.err.println(x: "Cambio exitoso!");  
        }  
        recorrer = recorrer.siguiente;  
    }  
} else {  
    System.out.println(x: " El beneficiario no encontrado!");  
}
```

Selección del patrón de diseño y justificación

Un patrón de diseño de software proporciona una estructura predefinida que ayuda a resolver de manera eficiente los problemas que se presentan en el desarrollo de una aplicación.

La selección del patrón MVC permite trabajar con aplicaciones que disponen de una estructura clara y organizada. Pensando en el sistema final, es fundamental contar con una metodología que facilite la gestión, el mantenimiento y la escalabilidad.

Esta división facilita el desarrollo y mantenimiento del proyecto, al evitar la mezcla de estas áreas críticas.

Modelo. Se trata del núcleo funcional que gestiona los datos manipulados en la aplicación.

Vista. Se trata de los componentes destinados a representar la información al usuario.

Controlador. Es un componente que recibe los eventos que provienen del usuario y los traduce en consultas para el modelo o para la vista.

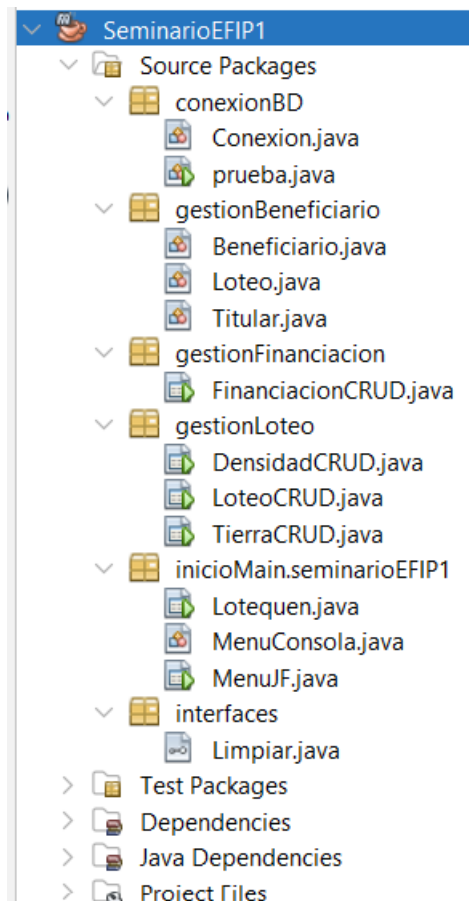
MVC es una forma de organizar el código de un sitio o una aplicación para ayudar a mantener el orden y estructura de un proyecto. Algunas justificaciones prácticas de por qué utilizar MVC serían, por ejemplo, las siguientes:

Si el cliente quiere cambiar los colores de la presentación o el color de un botón determinado, con MVC se tendrá que cambiar únicamente los archivos o recursos que estén implicados en la vista.

Si cambia algo en la base de datos o si agregamos cierta información en la estructura de una tabla, podremos realizar los cambios de sintaxis en los archivos correspondientes al modelo, sin temer que se hagan modificaciones involuntarias en el controlador.

En un MVC aplicado correctamente, la estructura del código se encuentra bien segmentada y ordenada.

Explicación del desarrollo, usando un patrón MVC



Inicialmente, quiero aclarar que la aplicación es muy simple, sin grandes aspiraciones visuales, por lo que no hay un paquete con imágenes. La segregación de paquetes la pensé en base a la cercanía de cada clase en la base de datos. Las clases Loteo, Tierra y Densidad tienen un alto grado de relación, por lo que están juntas, así, las clases Financiación o Beneficiario están separadas. Como dejo evidencia en la entrega 3 del trabajo integrador, la clase Beneficiario viene acompañada de su propia estructura de datos denominada *Titular*, con una muestra de lo que se podría hacer por consola para asignar y crear lotes y, de esta manera, agregar funcionalidad y cumplir con las estructuras solicitadas en dicha entrega. Inicialmente, constaba de una clase por cada “actividad” del CRUD, pero encontré la forma de hacerlo todo en una sola.

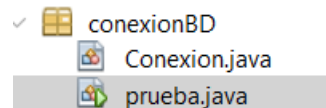
Otro paquete interesante es el de interfaces, al cual se le podrías agregar clases de apoyo con sobre carga de métodos, o clases abstractas de ser necesario. Más adelante voy a explicar porque decidí presentar la interfaz <Limpiar>.

Dada la importancia, le asigné un paquete propio a la conexión. Esta contiene la clase del mismo nombre con el método que será instanciado cada vez que se requiera la funcionalidad.

Por último, me parece que es una práctica agrupar por paquetes puntos neurálgicos en el flujo del programa, o al menos a mí me ayuda a identificarlos con claridad; de esta manera, la aplicación se

inicia por la clase denominada con el nombre del proyecto *Lotequen*, si el programa fuera más grande y requiriera más menús, se podría ver la necesidad de sumar paquetes; como no es el caso, desde el acceso el flujo se dirige a uno de los dos menús considerados como funcionalidades principales.

Persistencia y consulta de datos en una base de datos MySQL



La clase *Prueba*, la utilicé para poner a punto la conexión a la base de datos, antes de utilizar la misma desde las clases del sistema. En la entrega final a un cliente se eliminaría como parte de la depuración, ya que no tiene ninguna funcionalidad para este, pero es necesario tenerla para las pruebas finales y verificar si la conexión se realiza desde “ubicaciones diferentes”.

Establecer conexiones

```
public class Conexion {  
  
    private static final String DRIVER = "com.mysql.jdbc.Driver";  
    private static String USUARIO = "root";  
    private static String PASSWORD = "";  
    private static final String SCHEMA = "lotequen";  
    private static String URL = "jdbc:mysql://localhost:3306/" + SCHEMA;  
  
    static {  
        try {  
            Class.forName(className: DRIVER);  
        } catch (ClassNotFoundException e) {  
            System.out.println("Error en conexión local " + e);  
        }  
    }  
  
    public Connection getConnection() {  
  
        Connection con = null;  
  
        try {  
            con = DriverManager.getConnection(url: URL, user: USUARIO, password: PASSWORD);  
            /*JOptionPane.showMessageDialog(null, "Conexión exitosa");  
            lo utilicé para entender visulamente el estado de la conexión.*/  
        } catch (SQLException e) {  
            JOptionPane.showMessageDialog(parentComponent: null, "Error en conexión " + e);  
        }  
  
        return con;  
    }  
}
```

Realizar consultas

```
consulta = "select * from financiacion;";

try {
    st = cn.getConnection().createStatement();
    ResultSet rs = st.executeQuery(sql: consulta);

    while (rs.next()) {
        datosConsulta[0] = rs.getString(columnIndex: 1);
        datosConsulta[1] = rs.getString(columnIndex: 2);
        datosConsulta[2] = rs.getString(columnIndex: 3);
        datosConsulta[3] = rs.getString(columnIndex: 4);
        datosConsulta[4] = rs.getString(columnIndex: 5);

        modelo.addRow(rowData: datosConsulta);
    }
    tablaFinan.setModel(dataModel:modelo);
}
```

Las consultas a la BD se realizan con la palabra reservada *select*. Aquí se instancia la conexión, se abre la base de datos, se ejecuta la consulta, se recorre la respuesta alojando cada resultado en una ubicación del arreglo, columna por columna. Finalmente, se agrega cada fila a un modelo con el que se actualizará la tabla, para que pueda mostrarse mediante la interfaz.

Actualizar registros

```
valCuotaD = (((100 - apInicialD) / 100) / (añoI * 12));

try {
    Conexion cx = new Conexion();
    String update = "update financiacion set grupo = ?, años = ?,"
        + " aporte_inicial = ?, valor_cuota = ? where id_financiacion = ?;";
    CallableStatement cs = cx.getConnection().prepareCall(sql: update);

    cs.setString(parameterIndex: 1, x: grupoS);
    cs.setInt(parameterIndex: 2, x: añoI);
    cs.setDouble(parameterIndex: 3, x: apInicialD);
    cs.setDouble(parameterIndex: 4, x: valCuotaD);
    cs.setInt(parameterIndex: 5, x: indiceInt);
    cs.execute();

    LimpiarCasilleros();
    MostrarFinan(tablaFinan: jTable1);
    cs.close();
    JOptionPane.showMessageDialog(parentComponent: null, message: "Actualización exitosa!");
}
```

Las actualizaciones en la BD se realizan con la palabra reservada *update*. Aquí se instancia la conexión, se abre la base de datos y se ejecuta la consulta. Luego, se actualiza el contenido de la tabla para evidenciar el cambio en la lista. Finalmente, se limpian los casilleros y se da un cartel de actualización

exitosa para mejorar la comunicación con el usuario. Finalmente, se agrega cada fila a un modelo con el que se actualizará la tabla, para que pueda mostrarse mediante la interfaz.

Presentar resultados en la interfaz

The screenshot shows a web application window titled 'Tierra'. On the left, there is a form with the following fields and buttons:

- Regresar** (button)
- Descripcion Catastro:**
- Nombre:**
- Cantidad de Parcelas:**
- Crear Registro** (button)
- Modificar** (button)
- Eliminar** (button)
- Índice**

On the right, there is a table with the following data:

ID	Parcela	Catastro	Nombre
1	255	DRT54654	H8
3	95	5464fdg	K7
6	10	sfgasg4646	P1

At the bottom right, there is a link labeled **Acceso**.

Los resultados se pueden ver en la tabla adjunta. La misma se actualiza automáticamente al iniciar la clase, desde aquí se puede ver en detalle seleccionando el registro, o modificarlo, o eliminarlo, o crear uno nuevo si se cumplen las condiciones como ser un nombre no utilizado.

The screenshot shows the same web application window as before, but with some changes and annotations:

- The **Nombre:** field is now empty and highlighted with a green background.
- The **Cantidad de Parcelas:** field is now empty.
- The **Índice** field is now empty.
- The table has been updated with new data:

ID	Parcela	Catastro	Nombre
1	255	DRT54654	H8
3	95	5464fdg	K7
6	22	sfgasg4646	P1
7	9	546f5g4	srg

Handwritten annotations in red and green ink are present:

- A red circle around the ID '7' in the table.
- The word **NUOVO** (misspelled 'NUEVO') written in red below the table.
- The word **MODIFICADO** written in green below the table.

At the bottom right, there is a link labeled **Acceso**.

Correcta aplicación de excepciones para la interacción con la base de datos
MySQL

Cito dos casos a modo de ejemplo.

```

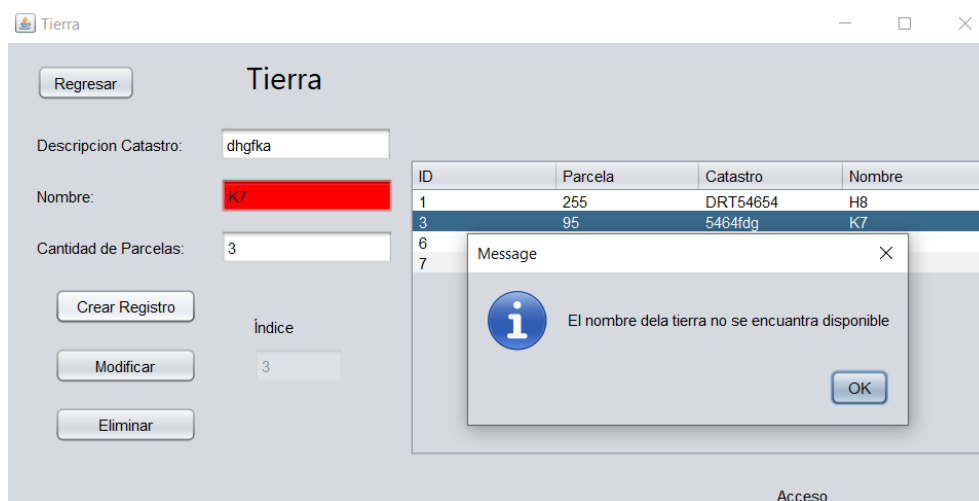
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent: null, message: "No se pueden mostrar registros: Tabla de Lote");
}

JOptionPane.showMessageDialog(parentComponent: null, message: "Actualización exitosa!");
} catch (SQLException e) {
    System.err.println("Error en la modificación del financiación." + e);
    JOptionPane.showMessageDialog(parentComponent: null, message: "Error, contacte al administrador (financiacion) !");
}

```

La conexión a base de datos tiene su propia excepción: *SQLException*. De todas formas, es conveniente sumar un texto particular a cada sección de código que de referencia al desarrollador de donde se origina el error, para poder encontrarlo con mayor facilidad y posteriormente corregirlo. Así mismo, si algo sale más es conveniente informar al usuario del error para tener un mejor feedback. Por lo que si la excepción saltó por alguna actividad del usuario, además, emerge un cartel de la interfaz para notificarlo.

A continuación, un cartel para informar de las reglas contempladas en los requisitos funcionales a los usuarios.



Inclusión pertinente de clases abstractas o interfaces

```

public class LoteoCRUD extends javax.swing.JFrame implements Limpiar {

```

Las clases desarrolladas con los JFrame, “extienden” a *javax.swing.JFrame*, por lo que no podrías realizar otra “extensión”, ya que Java solo permite un padre por hijo. Es decir, un padre puede tener muchos hijos, pero un hijo, no puede tener muchos padres. Aquí propongo aprovechar otras fortalezas

de la programación orientada a objetos, “implementando” interfaces. Sigo aprovechando el encapsulamiento y la herencia, mejorando la comprensión del código y facilitando el mantenimiento.

```
public interface Limpiar {  
  
    static final String VACIAR = "";  
  
    public void LimpiarCasilleros();  
}  
  
@Override  
public void LimpiarCasilleros() {  
    mzJT.setText(t: VACIAR);  
    csJT.setText(t: VACIAR);  
    idLotJT.setText(t: VACIAR);  
    idTierJT.setText(t: VACIAR);  
    idDenJT.setText(t: VACIAR);  
    nomenJT.setText(t: VACIAR);  
    densidadJT.setText(t: VACIAR);  
}
```

Nota: No es estrictamente necesario usar la anotación **@Override** cuando implementas un método de una interfaz, pero es una buena práctica hacerlo. Esta anotación indica que el método está sobrescribiendo un método de una superclase o implementando un método de una interfaz, lo que ayuda a evitar errores y mejora la legibilidad del código

Utilización complementaria de arreglos y de la clase ArrayList

Un poco de este contenido ya se fue adelantando en las explicaciones anteriores, todas las tablas “van guardando” el resultado de la consulta *select* en arreglos que se “van acomodando” mediante filas en una tabla. Todos los métodos *Mostrar** utilizan esta herramienta.

```
public void MostrarDensidad(JTable tablaDensidad) {

    this.jTable1 = tablaDensidad;

    Conexion cn = new Conexion();
    String consulta = "";
    DefaultTableModel modelo = new DefaultTableModel();
    String[] datosConsulta = new String[4];
    Statement st;
```

Además, para la entrega tres hice a mi modo de ver, una muy linda matriz de boléanos; que trabaja sin la conexión a base de datos, para saber si un lote fue ocupado o está vacío.

```
public class Loteo {

    private final int mz, cs;
    private final Boolean[][] loteo;
    private final Boolean vacio = true;
    private final Boolean ocupado = false;
```

Al resolver esto desempolvando viejos códigos escritos para las materias de Tallere de algoritmos, recordé como conectar mis propias estructuras de datos, por lo que no fue necesario apelar a las librerías *List* de Java. Para las codificaciones en JFrame, las listas no son necesarias ya que la información se va guardando para ser manipulada y ofrecida al usuario en tablas.

→ uso de librerías para tablas

```
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;
```

La librería **Sorter* permite ordenar la tabla por cualquiera de sus columnas en orden ascendente o descendente.

ID	Parcela	Catastro	Nombre ▲
1	255	DRT54654	H8
3	95	5464fdg	K7
6	22	sfgasg4646	P1
7	5	546f5g4	srg

Emplear archivos para guardar y recuperar información relevante

ID	Gupo	Años	Valor Inicial	Valor Cuota
12	D6	4	70.0	0.0062499...
14	D9	10	30.0	0.0058333...

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String archivo = "Financiacion.txt";
        BufferedWriter bw = new BufferedWriter(new FileWriter(fileName: archivo));
        for(int i=0; i<jTable1.getRowCount();i++){
            for(int j=0; j<jTable1.getColumnCount(); j++){
                bw.write((String) (jTable1.getValueAt(row: i, column: j)));
                if (j<jTable1.getColumnCount()-1){
                    bw.write(str: "|");
                }
            }
            bw.newLine();
        }
        bw.close();
    } catch(IOException e){
        JOptionPane.showMessageDialog(parentComponent: null, message: e);
    }
    // TODO add your handling code here:
}
```

Guarda un archivo con la tabla actual. No está dentro de los requisitos funcionales, aunque podría configurarse para guardar en .pdf y ser compartida a beneficiarios, o para imprimir y ofrecer si se acercan personalmente, como parte de la información relevante del loteo.

Presentación del proyecto en un video con una duración, aproximadamente, de 3 minutos

→ Lamentablemente la calidad del video y el audio no es la mejor. No tengo la placa de video adecuada para grabar pantalla en Windows; y en la Distribución Ubuntu no pude configurar el audio. Realicé una filmación externa a la pantalla con la cámara web.

Mas allá de la calidad, tres minutos para un video no es mucho por lo que espero haber contemplado las características principales de lo pretendido por la consigna.

GLOSARIO

Desarrollo territorial (DLLT): Son los usuarios finales del sistema a desarrollar.

Beneficiario: Personas físicas, aspirantes al plan de loteo social; o quienes accedieron al mismo.

Legajo: Carpeta física con la documentación presentada por el beneficiario.

Pliego: Es la documentación técnica y legal de la obra.

Datos: Son los atributos (o valores) que incluyen desde nombre y apellido, hasta precio de la cuota por un lote particular; pasando por la nomenclatura de los terrenos.

Ingreso: Es el dinero que puede justificar mediante recibos durante un mes. Se puede utilizar un promedio de un período determinado para asignar el valor mensual.

Tipo de loteo: Está dado por la ubicación, el costo de la tierra y el costo del loteo.

Densidad del lote: Está dado por el tamaño, las posibilidades de edificación y el uso (domiciliario, comercial, industrial, etc.)

Grupo: Los beneficiarios se segmentan en grupos según su ingreso, pudiendo acceder a diferentes tipos de financiación.