UNIVERSIDAD CATÓLICA DEL NORTE
Facultad de Ingeniería y Ciencias Geológicas
Departamento de Ingeniería de Sistemas y Computación

Programación Avanzada
Taller N°3

Integrantes del equipo:

- Stefant Contreras Rojas, 21.010.710-K (stefant.contreras@alumnos.ucn.cl)
- Felipe Gaete Fernandez, 20.416.511-4 (felipe.gaete@alumnos.ucn.cl)

Profesor:

- Tomás Reimann - (tomas.reimann@ce.ucn.cl)

Ayudantes:

- Rodrigo Aguilera - (rodrigo.aguilera01@alumnos.ucn.cl)
- Diego Cortés (diego.cortes07@alumnos.ucn.cl)

Antofagasta, 18 de Mayo de 2025

Introducción

En este informe inicial se tiene como objetivo explicar el diagrama de clases y los contratos realizados y solicitados para el taller 3 que es principalmente sobre Interfaces y Herencia. En este informe se describen los conceptos del sistema, sus relaciones y la estructura propuesta para la solución del taller.

Contexto sobre de qué se trata el programa y como soluciona el problema

En este taller se nos dice que la Universidad Católica del Norte (UCN) busca automatizar el proceso de creación y postulación a ofertas académicas. Para resolver la problemática entregada se requiere un sistema que nos permita registrar y gestionar diferentes tipos de ofertas académicas, así como los usuarios que interactúan con estas ofertas.

Dentro de nuestra solución tendríamos que crear estas funcionalidades para cumplir con los requerimientos que nos piden dentro del taller para crear el programa solicitado:

1. Una funcionalidad de registro e inicio de sesión de usuarios donde cada usuario tiene nombre, RUT, correo y contraseña, usando el RUT como identificador principal para acceder al sistema.
2. Una funcionalidad para ingresar ofertas académicas de tres tipos distintos (ayudantías, capstone y prácticas pre-profesionales), cada una con sus propios datos específicos además de la información común que comparten.
3. Una funcionalidad para buscar ofertas específicas a través de su código único o por su título, mostrando toda la información relevante al encontrarla.
4. Una funcionalidad para ver todas las ofertas registradas en el sistema de forma clara y ordenada.
5. Una funcionalidad para editar los datos de las ofertas existentes, buscándolas por su ID y permitiendo modificar todos los campos correspondientes.
6. Una funcionalidad para eliminar ofertas del registro usando su código único como identificador.
7. Una funcionalidad para que el usuario pueda consultar y modificar su información personal dentro del sistema.
8. Una funcionalidad para cerrar la sesión del usuario y volver al menú de registro inicial.

El sistema debe realizar validaciones en todos los datos ingresados, garantizando que cumplan con los requisitos específicos establecidos para cada tipo de oferta y usuario, además de leer inicialmente la información desde un archivo de texto con datos ya definidos.

Contratos Actualizados

1. registrarUsuario()

Operación	**registrarUsuario(nombre: String, rut: String, correo: String, contrasenia: String): boolean**
Descripción	Registra un nuevo usuario en el sistema con validaciones de integridad.
Pre-Condiciones	El nombre debe tener entre 3 y 255 caracteres. El RUT debe ser válido y único. El correo debe ser válido y único. La contraseña debe tener entre 6 y

255 caracteres.

Post-Condiciones	Se crea un nuevo usuario en el sistema. Retorna true si el registro es exitoso, false en caso contrario.

2. iniciarSesion()

Operación	**iniciarSesion(rut: String, contrasenia: String): boolean**
Descripción	Autentica a un usuario en el sistema utilizando RUT y contraseña.
Pre-Condiciones	El RUT debe existir en el sistema. La contraseña debe coincidir.
Post-Condiciones	Se establece al usuario como usuarioActual. Retorna true si es exitoso.

3. ingresarOferta()

Operación	**ingresarOferta(): void**
Descripción	Permite crear una nueva oferta académica con todos sus datos requeridos.
Pre-Condiciones	Debe existir un usuario conectado. Los datos deben cumplir validaciones específicas. El ID debe ser único.
Post-Condiciones	Se agrega una nueva oferta al contenedor con fechas calculadas automáticamente.

4. buscarOferta()

Operación	**buscarOferta(): void**

Descripción	Permite consultar una oferta específica por código único o título.
Pre-Condiciones	Debe existir un usuario conectado. El criterio de búsqueda debe ser válido.
Post-Condiciones	Si existe, se muestra toda la información. Si no existe, se notifica al usuario.

5. verOfertas()

Operación	**verOfertas(): void**
Descripción	Muestra el listado completo de ofertas académicas registradas.
Pre-Condiciones	Debe existir un usuario conectado en el sistema.
Post-Condiciones	Se despliega un listado con información resumida de todas las ofertas.

6. editarOferta()

Operación	**editarOferta(): void**
Descripción	Permite modificar los datos de una oferta académica existente.
Pre-Condiciones	Debe existir un usuario conectado. Debe existir la oferta a editar. Los nuevos datos deben ser válidos.
Post-Condiciones	Los datos de la oferta son reemplazados completamente en el sistema.

7. eliminarOferta()

Operación	**eliminarOferta(): void**
Descripción	Permite eliminar una oferta académica del registro.
Pre-Condiciones	Debe existir un usuario conectado. Debe existir la oferta. El usuario debe confirmar la eliminación.
Post-Condiciones	La oferta es eliminada del contenedor y reorganizada la estructura.

8. verPerfil()

Operación	**verPerfil(): void**
Descripción	Muestra los datos del usuario conectado excluyendo la contraseña.
Pre-Condiciones	Debe existir un usuario conectado en el sistema.
Post-Condiciones	Se muestra la información del usuario actual.

9. editarPerfil()

Operación	**editarPerfil(): void**
Descripción	Permite modificar información personal del usuario (nombre y correo).
Pre-Condiciones	Debe existir un usuario conectado. Los nuevos datos deben cumplir validaciones.
Post-Condicion	La información del usuario es actualizada. El RUT no es modificable.

es	

10. cerrarSesion()

Operación	**cerrarSesion(): void**
Descripción	Finaliza la sesión del usuario actual y vuelve al menú inicial.
Pre-Condiciones	Debe existir un usuario conectado en el sistema.
Post-Condiciones	El usuarioActual se establece como null. Se muestra el menú de registro.

Diagrama de Clases Actualizado

1. Usuario

Representa: Un usuario registrado en el sistema que puede gestionar ofertas académicas.

Atributos:

nombre: String - Nombre completo del usuario

rut: String - RUT del usuario (identificador único)

correo: String - Correo electrónico del usuario

contrasenia: String - Contraseña para autenticación

Métodos principales:

Usuario(nombre, rut, correo, contrasenia) - Constructor

gets() y sets() - Métodos de acceso a atributos

verificarContrasenia(contrasenia): boolean - Valida la contraseña

toString(): String - Representación textual del usuario

2. Oferta (Abstracta)

Representa: La clase base abstracta para todos los tipos de ofertas académicas.

Atributos:

id: String - Identificador único de la oferta

titulo: String - Título descriptivo de la oferta

descripcion: String - Descripción detallada de la oferta

nombreUnidad: String - Nombre de la unidad académica

duracionDias: int - Duración de la oferta en días

fechaPublicacion: LocalDate - Fecha de publicación (calculada automáticamente)

fechaCierre: LocalDate - Fecha de cierre (calculada automáticamente)

Métodos principales:

Oferta(id, titulo, descripcion, nombreUnidad, duracionDias) - Constructor

gets() y sets() - Métodos de acceso

calcularFechas(): void - Calcula fechas de publicación y cierre

toString(): String - Método abstracto implementado por subclases

3. Ayudantia extends Oferta

Representa: Una oferta para que estudiantes trabajen como ayudantes en asignaturas específicas.

Atributos específicos:

nombreAsignatura: String - Nombre de la asignatura

rolAyudante: String - Rol específico del ayudante

horasSemanales: int - Cantidad de horas semanales

promedioMinimo: double - Promedio mínimo requerido

tipoAyudantia: String - Tipo de ayudantía

Métodos principales:

Ayudantia(...) - Constructor con parámetros específicos

gets() y sets() para atributos específicos

toString(): String - Implementación específica

4. ActividadEmpresa extends Oferta (Abstracta)

Representa: Clase abstracta que agrupa características comunes de ofertas vinculadas a empresas.

Atributos específicos:

fechaInicio: LocalDate - Fecha de inicio de la actividad

nombreEmpresa: String - Nombre de la empresa solicitante

nombreGuia: String - Nombre del profesor guía

Métodos principales:

ActividadEmpresa(...) - Constructor

gets() y sets() para atributos específicos

5. Capstone extends ActividadEmpresa

Representa: Proyectos integradores que constituyen el trabajo final de titulación.

Atributos específicos:

tipoProyecto: String - Tipo de proyecto (Capstone, Investigación, Memoria)

duracionMeses: int - Duración del proyecto en meses

carrerasNecesarias: String[] - Carreras que pueden postular

cantidadMinEstudiantes: int - Cantidad mínima de estudiantes

Métodos principales:

Capstone(...) - Constructor completo

gets() y sets() para atributos específicos

toString(): String

6. Practica extends ActividadEmpresa

Representa: Instancias laborales para adquirir experiencia profesional.

Atributos específicos:

poseeRemuneracion: boolean - Indica si la práctica es remunerada

Métodos principales:

Practica(...) - Constructor completo

isPoseeRemuneracion() y setPoseeRemuneracion() - Métodos de acceso

toString(): String - Implementación específica

7. ContenedorOfertas

Representa: Una colección que gestiona todas las ofertas académicas del sistema.

Atributos:

ofertas: Oferta[] - Lista de ofertas

cantidadMaxima: int - Capacidad máxima del contenedor

cantidadActual: int - Número actual de ofertas

Métodos principales:

ContenedorOfertas(cantidadMaxima) - Constructor

getCantidadActual(): int - Retorna cantidad actual

agregar(nueva: Oferta): void - Añade nueva oferta

buscarPorId(id: String): int - Busca por ID, retorna posición

buscarPorTitulo(titulo: String): int - Busca por título parcial

obtener(posicion: int): Oferta - Recupera oferta por posición

eliminar(id: String): boolean - Elimina y reorganiza la lista

8. SistemaOfertas (Interface)

Representa: Contrato que define las operaciones principales del sistema.

Métodos:

registrarUsuario(String, String, String, String): boolean

iniciarSesion(String, String): boolean

ingresarOferta(): void

buscarOferta(): void

verOfertas(): void

editarOferta(): void

eliminarOferta(): void

verPerfil(): void

editarPerfil(): void

cerrarSesion(): void

9. SistemaOfertasUCN implements SistemaOfertas

Representa: Implementación concreta del sistema de gestión de ofertas.

Atributos:

contenedorOfertas: ContenedorOfertas - Gestiona todas las ofertas

usuarioActual: Usuario - Usuario con sesión activa

usuarios: Usuario[] - Lista de usuarios registrados

cantidadUsuarios: int - Contador de usuarios

archivoOfertas: String - Ruta del archivo de inicialización

Métodos principales:

SistemaOfertasUCN() - Constructor que inicializa y carga datos

leerArchivoOfertas(): void - Lector de archivo para carga inicial

Implementación de todos los métodos de la interfaz

getUsuarioActual(): Usuario - Método adicional

10. VistaConsola

Representa: Interfaz de usuario que maneja la interacción por consola.

Atributos:

sistema: SistemaOfertas - Referencia a la interfaz del sistema

sistemaUCN: SistemaOfertasUCN - Referencia específica

Métodos principales:

VistaConsola(sistema) - Constructor

iniciar(): void - Inicia la aplicación

menuRegistroLogin(): void - Gestiona registro e inicio de sesión

menuPrincipal(): void - Gestiona el menú principal

registrarUsuario(): void y iniciarSesion(): void - Interfaces específicas

lecturaEnteros(frase: String): int - Método auxiliar

11. ValidadorEntrada

Representa: Clase utilitaria con métodos estáticos para validación de datos.

Métodos principales:

validarEnteros(mensaje: String): int - Validación de enteros

validarDouble(mensaje: String): double - Validación de decimales

validarString(mensaje: String): String - Validación de strings

validarRut(rut: String): boolean - Algoritmo de validación de RUT chileno

validarCorreo(correo: String): boolean - Validación de email

validarTexto(texto: String, min: int, max: int): boolean - Validación de longitud

Métodos específicos para cada tipo de oferta

Relaciones y Multiplicidades

VistaConsola "1" --> "1" SistemaOfertas: La vista utiliza la interfaz del sistema

SistemaOfertasUCN ..|> SistemaOfertas: El sistema implementa la interfaz

SistemaOfertasUCN "1" o-- "1" ContenedorOfertas: El sistema tiene un contenedor

ContenedorOfertas o-- "*" Oferta: El contenedor gestiona múltiples ofertas

Usuario "1" -- "*" Oferta: Un usuario puede gestionar múltiples ofertas

Ayudantia --|> Oferta: Ayudantía hereda de Oferta

ActividadEmpresa --|> Oferta: ActividadEmpresa hereda de Oferta

Capstone --|> ActividadEmpresa: Capstone hereda de ActividadEmpresa

Practica --|> ActividadEmpresa: Práctica hereda de ActividadEmpresa

Descripción y Explicación del Código Realizado

Aplicación de Herencia, el sistema implementa una jerarquía de herencia.

Clase base abstracta Oferta: Define las características comunes a todas las ofertas (id, título, descripción, nombreUnidad, duracionDias) y maneja automáticamente el cálculo de fechas de publicación y cierre mediante el método calcularFechas().

ActividadEmpresa: Extiende de Oferta y añade atributos específicos para actividades que involucran empresas (fechaInicio, nombreEmpresa, nombreGuia). Esta clase actúa como una especialización que agrupa características comunes entre Capstone y Prácticas.

Implementaciones concretas:

Ayudantia hereda directamente de Oferta y añade atributos específicos como nombreAsignatura, rolAyudante, horasSemanales, promedioMinimo y tipoAyudantia.

Capstone y Practica heredan de ActividadEmpresa, aprovechando los atributos de empresa y añadiendo sus propias características específicas.

Uso de Interfaces

La interfaz SistemaOfertas define el contrato que debe cumplir cualquier implementación del sistema de gestión de ofertas. Esto permite: La posibilidad de crear diferentes implementaciones del sistema sin afectar el código cliente, realizar cambios en la implementación no afectan las clases que dependen de la interfaz y a la vez facilita la creación de implementaciones de borrador para pruebas.

Polimorfismo en Acción

El sistema utiliza polimorfismo principalmente en el ContenedorOfertas, que maneja objetos de tipo Oferta sin conocer su implementación específica. El método toString() es implementado de manera diferente en cada subclase:

- Ayudantia muestra información específica de ayudantías
- Capstone incluye información del proyecto y carreras necesarias
- Practica indica si posee remuneración

Encapsulación y Validaciones

Todas las clases mantienen sus atributos privados y proporcionan métodos getter/setter para el acceso controlado. El sistema incluye validaciones específicas implementadas en la clase ValidadorEntrada:

Validaciones generales: La validación de RUT chileno usando el algoritmo de módulo 11, la validación de formato de correo electrónico y la validación de longitud de strings (nombres, títulos, descripciones)

Validaciones específicas por tipo de oferta:

- **Ayudantías**: Promedio entre 4.0 y 7.0, horas semanales mayor a 2, tipo de ayudantía específico
- **Capstone**: Duración en meses (6, 12 o 18), cantidad de estudiantes entre 2 y 5, tipo de proyecto válido
- **Prácticas**: Validación de remuneración como valor booleano

Gestión de Datos y Contenedores

El ContenedorOfertas implementa una estructura de datos eficiente que permite:

- **Búsqueda por ID**: Comparación exacta de identificadores únicos
- **Búsqueda por título**: Búsqueda parcial case-insensitive usando contains()
- **Eliminación con reorganización**: Mantiene la integridad de la lista eliminando espacios vacíos
- **Control de capacidad**: Evita desbordamientos controlando la cantidad máxima

Lectura e Inicialización de Datos

Tenemos un sistema de lectura de archivo de texto (OfertasAcademicas.txt) que:

Lee líneas separadas por comas

Identifica el tipo de oferta según el primer carácter ('A', 'C', 'P')

Crea objetos específicos según el tipo identificado

Maneja fechas en formato "dd-MM-yyyy" usando DateTimeFormatter

Gestiona listas de strings para carreras en proyectos Capstone

Sistema de Autenticación

La gestión de usuarios implementa:

Registro: Validación completa de datos y verificación de unicidad de RUT y correo

Autenticación: Verificación de credenciales con método verificarContrasenia()

Sesión: Mantenimiento del estado del usuario actual durante la ejecución

Seguridad básica: Almacenamiento de contraseñas (aunque en texto plano para simplicidad académica)

Interfaz de Usuario por Consola

La clase VistaConsola implementa un sistema de menús jerárquicos que permite:

Navegación intuitiva: Menús numerados con opciones claras

Validación de entrada: Uso de ValidadorEntrada para asegurar entradas válidas

Manejo de errores: Reintentos automáticos cuando se ingresan datos inválidos

Separación de responsabilidades: La vista solo maneja presentación, delegando lógica al sistema

Tiempo Dedicado al Taller

Diseño y planificación: 5 horas
Realización de los diagramas: 5-6 horas *(+ 2 horas de actualizaciones para la entrega final)*
Codificación: 30-40 horas
Elaboración del informe: 6 horas - Stefant Contreras
Debugging y detección de errores: 7 horas
JavaDoc: 6 horas - Felipe Gaete