Proyecto 2 de Programación 1: Simulaciones con estructuras dinámicas y GUI

Fecha de Presentación:

Viernes 4 de Julio 2025, Hora de clase.

Fecha de Entrega:

Viernes 4 de Julio 2025 11:59PM

Objetivos:

1. **Desarrollar competencias en programación orientada a objetos aplicando estructuras de datos dinámicas** implementadas desde cero, como listas enlazadas y árboles binarios de búsqueda.

2. **Aplicar recursividad** en el manejo de árboles binarios para recorrer, insertar y buscar información.

3. **Integrar estructuras dinámicas con interfaces gráficas** utilizando Java Swing, promoviendo la modularidad, claridad de diseño y validación visual de los datos.

4. **Fomentar el uso de herramientas colaborativas** como Git, y reforzar buenas prácticas de programación, documentación interna y externa.

Penalización por uso de Inteligencia Artificial:

El uso de herramientas de Inteligencia Artificial para generar la solución completa del problema está terminantemente prohibido. Si se detecta el uso indebido de IA para resolver total o parcialmente el proyecto, se asignará una calificación de 0. El objetivo es que el proyecto sea una evidencia auténtica de su aprendizaje.

Planteamiento del Problema

Deberá implementar dos estructuras de datos desde cero: una lista enlazada (simple o doblemente enlazada) y un árbol binario de búsqueda. Luego, deberá escoger una de las siguientes simulaciones para construir un sistema que utilice esas estructuras y permita la visualización de la información mediante una interfaz gráfica con Java Swing:

1. Clínica Veterinaria:
    a. Estructuras requeridas:
        i. Lista enlazada dinámica para representar la cola de espera de atención.
        ii. Árbol binario de búsqueda (ABB) para almacenar las mascotas registradas (ordenadas por nombre o ID).

    b. Características:
        i. Cuando una mascota llega, se registra (si no existe) y se agrega a la cola de espera.
        ii. El veterinario atiende mascotas en orden (FIFO) y puede consultar historial.
        iii. El árbol permite buscar mascotas por nombre, eliminar registros o mostrar un recorrido ordenado.
        iv. La interfaz gráfica muestra la cola actual, la mascota siendo atendida y permite interactuar con el sistema para el registro de mascotas, etc.
        v. Usar recursividad para recorrer el árbol, borrar nodos, y mostrar el listado ordenado en GUI.

2. Sistema de Entregas:
    a. Estructuras requeridas:
        i. Lista enlazada dinámica para representar las rutas de entrega (pueden agregarse y quitarse direcciones).
        ii. Árbol binario de paquetes organizado por prioridad o destinatario.

    b. Características:
        i. Cada camión tiene una lista enlazada de direcciones. Se pueden insertar o quitar direcciones durante la jornada.
        ii. Cada dirección puede tener varios paquetes, representados en un árbol ordenado por nombre del destinatario.
        iii. El GUI permite ver el camión en ruta, los paquetes pendientes por dirección, y un mapa lógico de recorrido.
        iv. El recorrido en el árbol de paquetes se hace recursivamente, mostrando datos en orden o preorden.

3. Gestor de Biblioteca:
    a. Estructuras requeridas:
        i. Lista enlazada para registrar el historial de préstamos de cada usuario.

ii. Árbol binario por categoría/autor/título para almacenar libros disponibles.

b. Características:
   i. Los usuarios pueden ver su historial (usando GUI y recorriendo la lista).
   ii. El árbol permite buscar libros disponibles con búsquedas exactas o por prefijo (recursivas).
   iii. El sistema debe manejar préstamos, devoluciones, y mostrar recomendaciones.
   iv. La interfaz visualiza el árbol de libros, los libros prestados y permite filtrado interactivo.

Solución Esperada

El sistema debe cumplir con lo siguiente:

- Implementación propia de la lista enlazada y el árbol binario de búsqueda (sin usar colecciones de Java).
- Mostrar mediante interfaz gráfica las funcionalidades del caso escogido utilizando las estructuras de datos previamente implementadas.
- Manejo de inserciones, búsquedas y eliminaciones sobre las estructuras.
- Interfaz gráfica funcional con Java Swing y control de errores (el sistema no debe caerse)

Persistencia de Datos y Manejo de Errores

Sin importar el caso de simulación escogido, el estado del sistema debe poder **guardarse en un archivo y cargarse nuevamente al iniciar el programa**, de modo que el sistema pueda continuar funcionando aún después de haber sido cerrado.

Esto implica que la información crítica del sistema (como el historial de mascotas y cola de atención, las rutas de entrega, los libros disponibles o los préstamos activos) debe **almacenarse en archivos de texto** utilizando clases como `File`, `FileWriter`, `FileReader`, `BufferedReader`, `PrintWriter`, u otras herramientas básicas de Java.

Además, es **obligatorio** utilizar bloques `try-catch` para manejar posibles errores de lectura, escritura, entradas inválidas del usuario y otros posibles errores. El sistema debe estar protegido contra caídas inesperadas: si ocurre un error en la carga de archivos o el ingreso de datos, debe mostrarse un mensaje al usuario, sin detener el funcionamiento general del programa.

Entradas y Salidas:

Utilice componentes de Java Swing como JTextFields, JButtons, JTextArea, JLabel, entre otros para capturar información y mostrar resultados. No se permite que el sistema opere por consola.

Documentación Interna:

Comente el código fuente explicando el propósito de cada clase y cada método. Arriba de cada método debe describirse:

- Los parámetros de entrada y su tipo.
- El valor de retorno, si aplica.
- Una breve descripción de lo que hace el método.
- Puntos extra (2.5pts): Si desea investigar y utilizar la biblioteca doxygen para generar los comentarios.

Documentación Externa:

- Archivo README.md: Incluir un archivo README.md que explique cómo ejecutar el proyecto, cómo usar y cualquier otro detalle relevante para el usuario. Puntos extra si es formato markdown (2.5pts).

- Documento de Decisiones: Crear un documento de texto adicional que detalle las siguientes secciones:

    a. Acuerdos de Pareja: Describir los acuerdos alcanzados entre los miembros del equipo, como la división del trabajo y las responsabilidades. Qué se acuerda en casos de desacuerdo o problemas, etc.

    b. Decisiones de Diseño: Explicar cómo se diseñaron las clases, cómo se estructuraron los métodos y cualquier decisión significativa tomada durante el diseño.

    c. Decisiones de Implementación: Detallar las decisiones específicas de implementación, como la elección de estructuras de datos, algoritmos utilizados, etc.

    d. Puntos de Mejora: Identificar áreas que podrían mejorarse o ampliarse en futuras versiones del proyecto.

Control de Versiones:

Deberán usar un repositorio Git para el control de versiones y colaborar en el desarrollo del proyecto. Cada pareja debe crear un repositorio en GitHub (usar el ya existente del curso) y realizar commits regularmente para documentar el progreso. De no registrarse un incremento gradual mediante commits se penalizará fuertemente en la nota (50%).

Entregables

Usted deberá entregar una carpeta comprimida en formato "`.zip`" con el "`carne1_carne2_proyecto1.zip`" en mediación virtual que tendrá:

1. Código Fuente: Todo el código fuente debe estar disponible en el repositorio de GitHub y entregado en la entrega respectiva vía mediación virtual. (deben coincidir las versiones de lo subido en github con lo entregado en la fecha de entrega)
2. Documentación Interna: Comentarios dentro del código.
3. Documentación Externa:
    a. Archivo README.md con instrucciones de uso y detalles relevantes.
    b. Documento de texto adicional con decisiones de diseño, acuerdos de pareja, decisiones de implementación y puntos de mejora.

¡Buena suerte con el proyecto! Si necesitan ayuda durante el desarrollo, no duden en preguntar.