Trabajo Práctico 1 de Java POO

Parte de Herencia y Polimorfismo

Herencia y Jerarquía de Clases

En este trabajo se aplicó la herencia para organizar las clases que representan a las personas dentro del sistema universitario. La clase principal es Persona, que contiene los atributos comunes como nombre, apellido, documento y edad. Luego, la clase Estudiante hereda de Persona y agrega sus propios atributos, como carrera y un array de materias. Esto permite reutilizar código y mantener una estructura más clara, ya que toda la información personal se define una sola vez. De esta forma, cuando se crean clases nuevas como Profesor o Personal, pueden heredar de Persona y aprovechar la misma base.

Polimorfismo e Interfaces

El polimorfismo (que significa “muchas formas”) se aplicó mediante una interfaz llamada MiembroUniversidad. Esta interfaz define métodos como obtenerRol() y obtenerInformacionCompleta(), que todas las clases deben implementar. Gracias a esto, las clases Estudiante, Profesor y Personal comparten un mismo formato, aunque cada una tenga su propia forma de responder. En la clase Universidad, todos los miembros se manejan con un array polimórfico del tipo MiembroUniversidad[]. Esto permite recorrer el array y ejecutar métodos comunes sin importar el tipo de objeto. Por ejemplo, al llamar a miembro.obtenerRol(), Java detecta automáticamente qué versión del método ejecutar.

2. Algoritmos Recursivos e Iterativos

En el proyecto se usaron dos formas distintas para resolver operaciones: una iterativa (usando bucles) y otra recursiva (usando llamadas al mismo método). Por ejemplo, el promedio de notas se calcula con una versión iterativa, mientras que las funciones de búsqueda y conteo tienen versiones tanto recursivas como iterativas.

El enfoque iterativo usa bucles como for o while. Es el más común para tareas lineales como calcular promedios o buscar datos. Tiene la ventaja de usar menos memoria y ejecutarse más rápido, aunque puede quedar un poco más largo. El enfoque recursivo se basa en que un método se llama a sí mismo hasta llegar a un caso base. Es más corto y fácil de leer, pero consume más memoria y puede causar errores si hay demasiadas llamadas.

3. Algoritmos de Ordenamiento y Búsqueda

Algoritmo de Ordenamiento
Para ordenar a los estudiantes por su apellido se implementó el algoritmo Selection Sort en el método Universidad.ordenarPorApellido(Estudiante[] estudiantes). Este algoritmo recorre la lista buscando el apellido más chico (alfabéticamente) y lo intercambia con el elemento actual. Aunque no es el más rápido en listas grandes, sirve para entender la lógica de ordenamiento y preparar el array para la búsqueda binaria.

Algoritmo de Búsqueda: Búsqueda Binaria
La búsqueda binaria se implementó en Universidad.busquedaBinariaEstudiante(Estudiante[] estudiantes, String apellido). Compara el apellido buscado con el del medio del array: si coincide lo encuentra, si es menor busca a la izquierda y si es mayor busca a la derecha. Este proceso se repite hasta hallar el elemento o agotar la lista. Es mucho más eficiente que la búsqueda lineal, con una complejidad de O(log n). Es importante que el array esté ordenado previamente para que funcione correctamente.