PROGRAMACIÓN II - COLECCIONES

Bienvenido a la unidad de Colecciones en Programación II. Esta microteaching explora estructuras de datos esenciales y técnicas avanzadas en Java para la gestión eficiente de la información.

Resultados de Aprendizaje

Gestión de Colecciones Dinámicas

Dominar la implementación y manipulación de estructuras de datos dinámicas con **ArrayList** en Java, aplicando métodos esenciales para una gestión eficiente de elementos.

Modelado de Relaciones 1 a N

Representar y programar relaciones **uno a muchos (1:N)** en diagramas UML e implementarlas en Java, asegurando encapsulación y adherencia a buenas prácticas de diseño.

Recorrido Eficiente de Colecciones

Aplicar y comparar el ciclo **for-each** para recorrer colecciones, diferenciándolo del bucle **for** tradicional en términos de legibilidad y eficiencia para el procesamiento de datos.

Algoritmos de Búsqueda y Ordenamiento

Desarrollar e implementar **algoritmos** para buscar, ordenar y filtrar datos en colecciones, optimizando la eficiencia en la manipulación de grandes volúmenes de información.

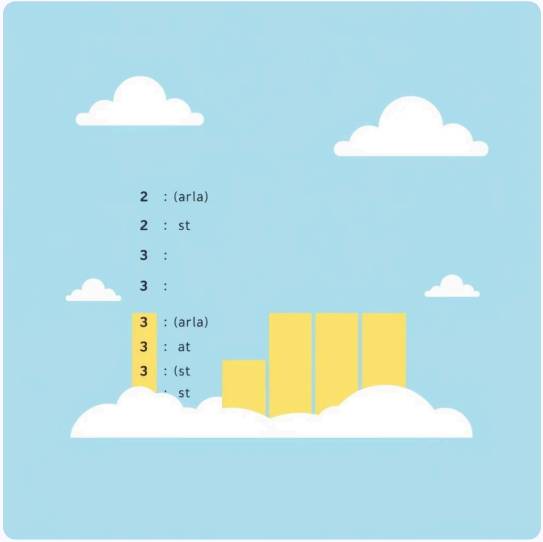
Definición y Uso de Enumeraciones

Utilizar y evaluar **enumeraciones (enums)** para representar conjuntos de valores predefinidos, mejorando la organización del código y la gestión de estados en aplicaciones robustas.

ArrayList en Java: La Lista Redimensionable

El **ArrayList** es una clase fundamental dentro del **Java Collections Framework**, actuando como una implementación redimensionable de la interfaz **List**. A diferencia de los arreglos tradicionales, su tamaño puede crecer o reducirse dinámicamente según sea necesario.

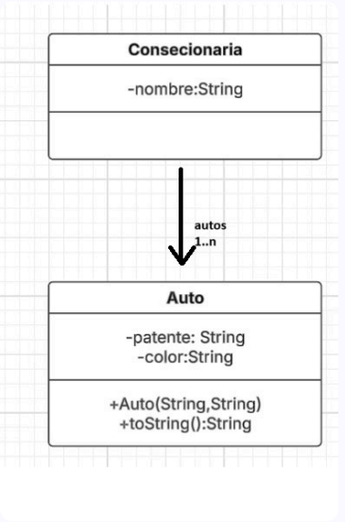
- Flexibilidad en el tamaño de los datos.
- Acceso rápido a elementos por índice.
- Facilita la adición y eliminación de elementos.
- Ideal para colecciones donde el número de elementos varía.

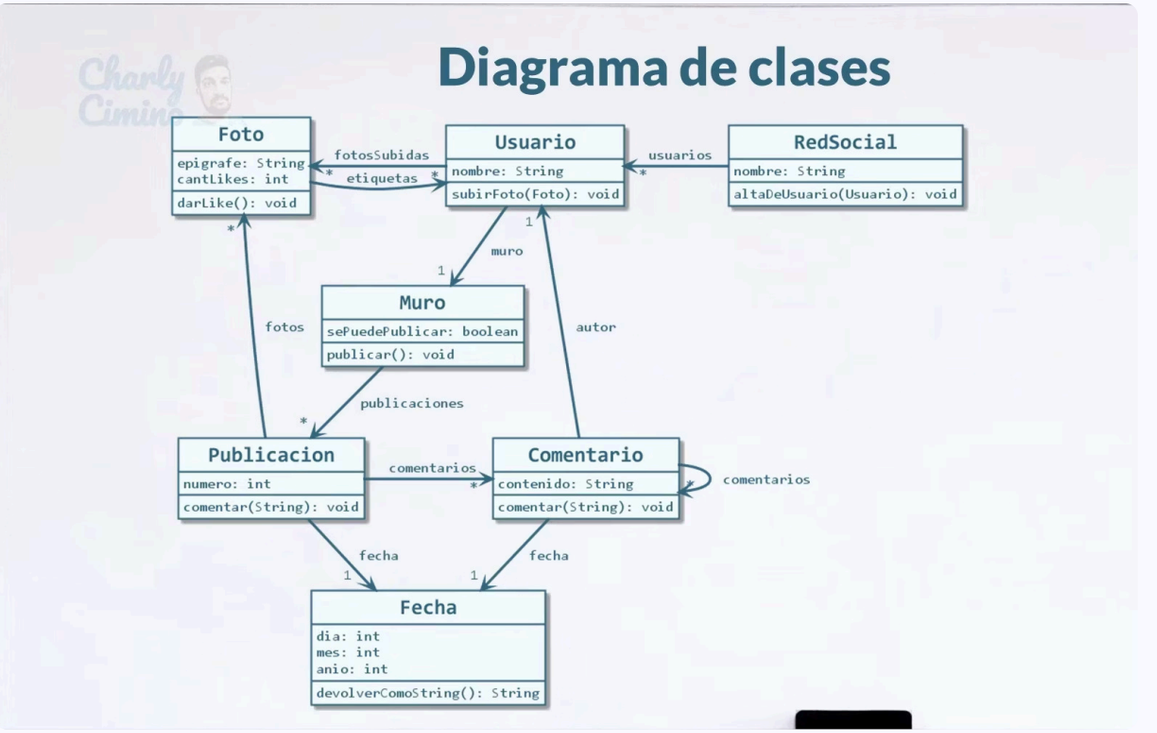


Relaciones Uno a Muchos (1:N)

Hasta ahora, hemos estudiado relaciones uno a uno. Sin embargo, en el mundo real, es común encontrar escenarios donde una entidad se relaciona con múltiples instancias de otra. La cardinalidad * (asterisco) denota "cero o muchos".

> "Una relación 1:N significa que una instancia de la primera entidad puede estar asociada con cero, una o muchas instancias de la segunda entidad."





Bucle For-Each: Recorriendo Colecciones con Elegancia

El bucle **for-each** simplifica la iteración sobre colecciones, ofreciendo una sintaxis más limpia y legible en comparación con el bucle **for** tradicional. Es especialmente útil cuando solo necesitamos acceder a los elementos, sin preocuparnos por sus índices.

Bucle For Clásico

```
List numeros = Arrays.asList(1,2,3,4,5);
for (int i = 0; i < numeros.size(); i++) {
    System.out.println(numeros.get(i));
}
```

Bucle For-Each

```
List numeros = Arrays.asList(1,2,3,4,5);
for (int numero : numeros) {
    System.out.println(numero);
}
```

La simplicidad del **for-each** reduce la probabilidad de errores de índice y mejora la claridad del código, siendo la opción preferida para iteraciones básicas.

Enums en Java: Constantes con Propósito

Las **enumeraciones (enums)** en Java son un tipo especial de clase que restringe la creación de objetos a un conjunto predefinido de valores. Son ideales para representar colecciones finitas y fijas de constantes, como los días de la semana, los meses o los estados de una aplicación.

Valores Constantes

Cada elemento de un **enum** es una instancia única y constante de la enumeración, lo que garantiza la seguridad del tipo y la previsibilidad.

Mejora la Legibilidad

Utilizar **enums** hace el código más claro y auto-documentado, reemplazando cadenas o números "mágicos" con nombres descriptivos.

Seguridad de Tipo

Evita errores de tiempo de ejecución al restringir los valores posibles, a diferencia de los **String** o **int** que pueden aceptar cualquier entrada.

Gestión de Estados

Perfectos para manejar estados en máquinas de estados o para definir opciones en interfaces de usuario.