POO - Java

jasson DB

December 2024

1 Introduction

Estas son notas del curso de Java dada por Sergie Code

2 Bases de la POO

2.1 Clases

Son la piedra angular de la POO actuando como **plantillas** para la creación de **objetos**. Definen las **propiedades** (atributos) y **comportamientos** (métodos) que caracterizan a los **objetos** de un tipo especifico.
Al encapsular datos y funcionalidades relacionadas, permiten la reutilización del código, la organización modular y la abstracción.

2.2 Atributos

Son las **características** o datos que describen el **estado** de un objeto en un contexto determinado. Estos pueden representar desde valores simples como números o cadenas de texto, hasta estructuras mas complejas como objetos u otros tipos de datos. Los **atributos** definen las **propiedades** de un objeto con los cuales se gestiona el **estado** del mismo.

2.3 Métodos

Son bloques de código que encapsulan una serie de acciones o **comportamientos** específicos asociados a un **objeto**, modificar su **estado** interno y responder a eventos dentro de un programa. Los **métodos** pueden acceder a los **atributos** de un **objeto**.

2.4 Objetos

Los **objetos** son **instancias** especificas de una **clase** que encapsulan **datos** y **comportamientos** relacionados.
Los **objetos** permiten modelar entidades del mundo real de manera modular y reutilizable, facilitando la organización y el diseño de sistemas de software eficientes y mantenibles.

3 Características de la POO

3.1 Abstracción

Permite representar entidades sel mundo real como **objetos** con **características** y **comportamientos** relevantes para el problema que se esta resolviendo.
Esto simplifica la complejidad del sistema al enfocarse en aspectos esenciales y proporcionar una forma de modelar conceptos del mundo real en el código.

3.2 Encapsulamiento

Los detalles internos de un **objeto** deben estar ocultos fuera de su definición y solo deben ser accesibles a través de una interfaz claramente definida.
Esto promueve la seguridad y la integridad de los datos al prevenir accesos no autorizados y facilita el mantenimiento del código. (EJM: Getters and Setters)
"Seria como pasar los atributos de una clase a a privado y solo puedan ser accedidos mediante los getter y setters"
getter (tomar información y devolverla).

3.3 Herencia

Permite que una **clase** (subclase) herede **atributos** y **métodos** de otra **clase** (superclase), lo que promueve la reutilización del código y la organización jerárquica de las clases.
Esto facilita la creación de nuevas clases que **extienden** el comportamiento de clases existentes, permitiendo una mayor flexibilidad y **modularidad** en el diseño del sistema.

3.4 Polimorfismo

Se refiere a la capacidad de **objetos** de diferentes clases de responder al mismo mensaje de manera diferente.
En otras palabras, un mismo **método** o mensaje puede producir diferentes resultados según el tipo de objeto que lo reciba. Esto permite escribir código mas genérico y re-utilizable.

4 Colecciones

Una **colección** es un **objeto** que agrupa **múltiples elementos** en una sola unidad. Estas colecciones proporcionan una forma mas conveniente de trabajar con **grupos de objetos** que las simples matrices(arrays) debido a su flexibilidad y funcionalidad adicional.

Figure 1: Colecciones

Característica	Set	List	Map
Orden	No hay garantía de orden.	Ordenado.	No hay garantía de orden en las claves, pero las claves-valor están asociadas.
Duplicados	No permite elementos duplicados.	Permite elementos duplicados.	No permite claves duplicadas, pero los valores pueden ser duplicados.
Implementaciones	HashSet, TreeSet, LinkedHashSet, etc.	ArrayList, LinkedList, Vector, etc.	HashMap, TreeMap, LinkedHashMap, etc.
Acceso por índice	No permite el acceso por índice.	Permite acceso por índice.	No permite acceso por índice de clave o valor.
Búsqueda rápida	Buena eficiencia en búsqueda de elementos.	La eficiencia depende de la implementación.	Buena eficiencia en búsqueda de claves, pero no de valores.
Uso	Útil para asegurar elementos únicos.	Útil para almacenar una secuencia de elementos donde el orden es importante.	Útil para almacenar asociaciones de claves-valor.
Ejemplo de uso	Eliminar duplicados en una lista.	Mantener una secuencia de elementos en orden.	Almacenar pares de datos relacionados.

4.1 SET: tipos de set

Una **colección** que no permite elementos **duplicados**. Se utiliza para almacenar elementos únicos sin ningún orden en particular.

Figure 2: Conjuntos

Característica	HashSet	TreeSet	LinkedHashSet
Implementación	Utiliza una tabla hash para almacenar los elementos.	Utiliza una estructura de árbol rojo-negro para almacenar los elementos, lo que garantiza un orden natural o utilizando un comparador personalizado.	Utiliza una combinación de tabla hash y lista doblemente enlazada para almacenar los elementos, manteniendo el orden de inserción.
Orden	No hay garantía de orden.	Ordenado según el criterio natural de los elementos o un comparador personalizado.	Mantiene el orden de inserción de los elementos.
Duplicados	No permite elementos duplicados.	No permite elementos duplicados.	No permite elementos duplicados.
Acceso por índice	No es posible acceder por índice.	No es posible acceder por índice.	No es posible acceder por índice.
Eficiencia en búsqueda	Búsqueda rápida en promedio, O(1) para operaciones de agregar, eliminar y comprobar si contiene.	Búsqueda más lenta debido a la estructura de árbol, O(log n) para operaciones de agregar, eliminar y comprobar si contiene.	Búsqueda rápida en promedio, O(1) para operaciones de agregar, eliminar y comprobar si contiene.
Uso común	Útil cuando no se necesita ningún orden particular y se requiere una búsqueda rápida.	Útil cuando se necesita un conjunto ordenado de elementos, ya sea naturalmente o por un criterio personalizado.	Útil cuando se necesita mantener el orden de inserción de los elementos y aún así se desea una búsqueda rápida.

4.2 LIST: tipos de List

Una **colección** ordenada de elementos que permite elementos duplicados. Los elementos en una lista están **indexados** y se pueden acceder por su **posición**.

Figure 3: Listas

Característica	ArrayList	LinkedList	Vector
Implementación	Utiliza un arreglo dinámico para almacenar los elementos.	Utiliza una lista doblemente enlazada para almacenar los elementos.	Utiliza un arreglo dinámico similar a ArrayList, pero es sincronizado, lo que garantiza seguridad en entornos multihilo.
Acceso por índice	Acceso rápido a los elementos por índice, O(1).	Acceso más lento a los elementos por índice debido a la necesidad de recorrer la lista, O(n).	Acceso rápido a los elementos por índice, O(1), similar a ArrayList.
Inserción y eliminación	Más lento para inserciones y eliminaciones en el medio de la lista debido a la necesidad de desplazar elementos, O(n).	Más rápido para inserciones y eliminaciones en el medio de la lista debido a la estructura de lista enlazada, O(1).	Similar a ArrayList en términos de rendimiento para inserciones y eliminaciones.
Eficiencia en memoria	Puede desperdiciar algo de memoria debido al tamaño fijo del arreglo interno.	Usa más memoria debido a los punteros adicionales en cada nodo.	Similar a ArrayList en términos de uso de memoria.
Iteración	Iteración rápida a través de los elementos, especialmente para acceder a través de bucles for.	Iteración más lenta debido a la necesidad de seguir enlaces de nodo a nodo.	Similar a ArrayList en términos de iteración.
Uso común	Útil cuando se necesita un acceso rápido a los elementos por índice y se realizan pocas inserciones y eliminaciones en el medio de la lista.	Útil cuando se realizan muchas inserciones y eliminaciones en el medio de la lista, o cuando se necesita una lista sincronizada para uso en entornos multihilo.	Menos comúnmente usado en comparación con ArrayList y LinkedList debido a su sincronización. Se usa en situaciones donde se requiere una lista segura para hilos.

4.3 MAP: tipos de maps

Una **colección** de **pares clave-valor** donde cada **clave** es **única**.

Se utiliza para almacenar **asociaciones de datos** donde cada **clave** esta asociada con un único valor.

No permite claves repetidas, pero los valores pueden ser repetidas.

Figure 4: Mapas

Característica	HashMap	TreeMap	LinkedHashMap
Implementación	Utiliza una tabla hash para almacenar las entradas.	Utiliza una estructura de árbol rojo-negro para almacenar las entradas, lo que garantiza un orden natural de las claves.	Combina una tabla hash con una lista doblemente enlazada para mantener el orden de inserción de las entradas.
Orden de las entradas	No hay garantía de orden en las entradas.	Las entradas están ordenadas según las claves, ya sea naturalmente o utilizando un comparador personalizado.	Mantiene el orden de inserción de las entradas.
Eficiencia en búsqueda	Búsqueda rápida en promedio, O(1) para operaciones de agregar, eliminar y obtener.	Búsqueda más lenta debido a la estructura de árbol, O(log n) para operaciones de agregar, eliminar y obtener.	Búsqueda rápida en promedio, O(1) para operaciones de agregar, eliminar y obtener.
Uso común	Útil cuando no se necesita ningún orden particular en las entradas y se requiere una búsqueda rápida.	Útil cuando se necesita un mapa ordenado según las claves, ya sea naturalmente o por un criterio personalizado.	Útil cuando se necesita mantener el orden de inserción de las entradas y aún así se desea una búsqueda rápida.
Iteración	Iteración rápida a través de las entradas, sin garantía de orden.	Iteración lenta debido a la necesidad de mantener el orden de las claves, pero el orden garantizado durante la iteración.	Iteración rápida a través de las entradas, manteniendo el orden de inserción.