LENGUAJE DE PROGRAMACION III

Trabajo Autónomo Grupal N°5

Profesor: Osvaldo Enrique Micniuk

Objetivos:

1. Comprender el funcionamiento y los métodos disponibles para diferentes tipos de colecciones en Java.
2. Analizar beneficios y desventajas de cada tipo de colección según casos de uso específicos.
3. Introducirse en el manejo de errores y excepciones en Java.

Integrantes

1° Edwar Meza

2°Abril Pereira

1. Investigación de Colecciones:

List

Descripción

Una List es una colección ordenada que permite elementos duplicados.

Métodos Clave

- **add()**: Agrega un elemento a la lista.
- **remove()**: Elimina un elemento de la lista.
- **get()**: Obtiene el elemento en una posición específica de la lista.

Usos Recomendados

Cuando se necesita una colección ordenada y elementos duplicados son permitidos, como por ejemplo, en una lista de tareas pendientes.

2. Set

Descripción

Un Set es una colección que no permite elementos duplicados.

Métodos Clave

- **add()**: Agrega un elemento al conjunto.
- **remove()**: Elimina un elemento del conjunto.

Usos Recomendados

Cuando se necesitan almacenar elementos únicos, como por ejemplo, en una lista de nombres de usuarios.

3. Map

Descripción

Un Map es una colección de pares clave-valor.

Métodos Clave

- **put()**: Asocia un valor a una clave en el mapa.
- **get()**: Obtiene el valor asociado a una clave en el mapa.
- **remove()**: Elimina el par clave-valor correspondiente a una clave en el mapa.

Usos Recomendados

Para asociar valores con claves únicas y realizar búsquedas rápidas por clave, como por ejemplo, en un diccionario.

4. **Queue**

Descripción

Una Queue es una colección que representa una cola (FIFO - First In, First Out).

Métodos Clave

- **offer()**: Agrega un elemento a la cola.
- **poll()**: Remueve y devuelve el elemento al principio de la cola.

Usos Recomendados

Cuando se necesita procesar elementos en orden de llegada, como por ejemplo, en un sistema de gestión de mensajes.

5. Stack

Descripción

Un Stack es una colección que representa una pila (LIFO - Last In, First Out).

Métodos Clave

- **push()**: Agrega un elemento a la pila.
- **pop()**: Remueve y devuelve el elemento en la cima de la pila.

Usos Recomendados

Cuando se necesita procesar elementos en el orden inverso al que fueron agregados, como por ejemplo, en la reversión de una cadena.

Cuadro Comparativo de Colecciones en Java

Colección	Descripción	Métodos Clave	Usos Recomendados
List	Colección ordenada que permite duplicados.	add(), remove(), get()	Cuando se necesita una colección ordenada.
Set	Colección que no permite duplicados.	add(), remove()	Cuando no se permiten elementos duplicados.
Map	Colección de pares clave-valor.	put(), get(), remove()	Para asociar valores con claves únicas.
Queue	Colección que representa una cola (FIFO).	offer(), poll()	Para procesar elementos en orden de llegada.
Stack	Colección que representa una pila (LIFO).	push(), pop()	Para procesar elementos en orden inverso al de llegada.

Análisis de Beneficios y Desventajas

Ventajas

- **List**: Garantiza el orden de inserción.
- **Set**: Garantiza la unicidad de elementos.
- **Map**: Permite asociar rápidamente valores con claves únicas.
- **Queue**: Soporta procesamiento FIFO.
- **Stack**: Soporta procesamiento LIFO.

Desventajas

- **List**: La búsqueda puede ser lenta en grandes listas.
- **Set**: No mantiene orden.
- **Map**: Uso ineficiente de memoria para claves duplicadas.
- **Queue**: Acceso lento a elementos intermedios.
- **Stack**: Puede causar desbordamiento de pila si se agregan demasiados elementos.

Investigación sobre Excepciones en Java

¿Qué es una excepción en Java y cómo se diferencia de un error?

Una excepción en Java es un evento que interrumpe el flujo normal de ejecución de un programa debido a una condición anormal, como un error de tiempo de ejecución. Se diferencia de un error en que las excepciones se pueden manejar y recuperar dentro del programa, mientras que los errores generalmente son condiciones irreparables.

Tipos de Excepciones

- **Excepciones chequeadas**: Aquellas que el compilador obliga a manejar o declarar en un bloque try-catch. Ejemplos incluyen IOException.
- **Excepciones no chequeadas**: Aquellas que el compilador no requiere manejar explícitamente. Ejemplos incluyen NullPointerException.
-

Uso de try, catch, finally, y throw

- **try**: Bloque de código donde se espera que ocurra una excepción.
- **catch**: Bloque de código que maneja una excepción.

- **finally**: Bloque de código que siempre se ejecuta, independientemente de si se produjo una excepción o no.
- **throw**: Palabra clave utilizada para lanzar una excepción explícitamente.

Ejemplos de Manejo de Excepciones

```
try {
    int resultado = 10 / 0; // Esto lanzará una ArithmeticException
} catch (ArithmeticException e) {
    System.out.println("Error aritmético: " + e.getMessage());
} finally {
    System.out.println("Finalizando...");
}
```