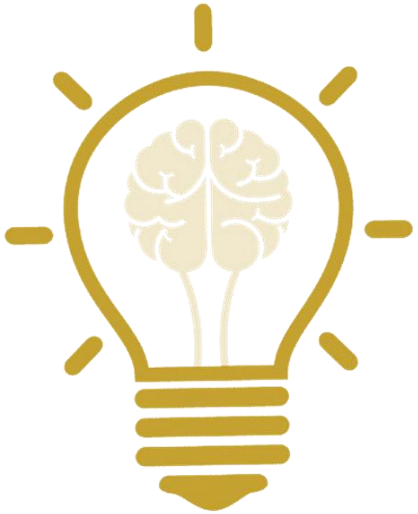
PROGRAMACIÓN EN JAVA

Materia PJA-001 material creado por el Ing. Jim Requena

Programación Orientada a Objetos en JAVA (Más sobre Colecciones en Java)



Materia PJA-001 material creado por el Ing. Jim Requena

Objetivos de la sesión

- Profundizar en el uso de colecciones avanzadas
- Aprender sobre Queue, Deque y sus implementaciones
- Entender y utilizar colecciones sincronizadas y no modificables
- Trabajar con la clase Collections para operaciones útiles
- Practicar con ejercicios avanzados de colecciones

Queue y Deque

Queue: Cola (FIFO - First In, First Out)

- Principales implementaciones: LinkedList, PriorityQueue

Deque: Cola de doble extremo

- Principales implementaciones: ArrayDeque, LinkedList

Ejemplo de Queue:

```java
Queue<String> cola = new LinkedList<>();
cola.offer("Primero");
cola.offer("Segundo");
cola.offer("Tercero");
System.out.println(cola.poll()); // Imprime: Primero
```

PriorityQueue

iTDS | Profesionales en Educación

- Implementa una cola de prioridad
- Los elementos se ordenan según su orden natural o un Comparator

Ejemplo

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.offer(5);
pq.offer(2);
pq.offer(8);
System.out.println(pq.poll()); // Imprime: 2
```

- El método poll() en una Queue (cola) en Java es una operación que se utiliza para recuperar y eliminar el elemento en la cabeza de la cola.
- En una PriorityQueue, poll() devuelve el elemento con la prioridad más alta.
- poll() es seguro y flexible para obtener y eliminar elementos de una cola, especialmente útil cuando no estámos seguros del estado de la cola y queremos evitar excepciones.

ArrayDeque

- Implementación de Deque basada en arreglos
- Más eficiente que Stack cuando se usa como pila
- Más eficiente que LinkedList cuando se usa como cola

Ejemplo

```java
Deque<String> deque = new ArrayDeque<>();
deque.addFirst("Primero");
deque.addLast("Último");
System.out.println(deque.removeFirst()); // Imprime: Primero
System.out.println(deque.removeLast()); // Imprime: Último
```

Colecciones sincronizadas

- Métodos de Collections para crear versiones thread-safe de colecciones

```
List<String> lista = Collections.synchronizedList(new ArrayList<>());
Set<Integer> conjunto = Collections.synchronizedSet(new HashSet<>());
Map<String, Integer> mapa = Collections.synchronizedMap(new HashMap<>());
```

Colecciones no modificables

- Métodos de Collections para crear versiones de solo lectura de colecciones

```
List<String> lista = Collections.unmodifiableList(Arrays.asList("A", "B", "C"));
Set<Integer> conjunto = Collections.unmodifiableSet(new HashSet<>(Arrays.asList(1, 2, 3)));
Map<String, Integer> mapa = Collections.unmodifiableMap(new HashMap<>());
```

Métodos útiles de Collections

- disjoint(Collection c1, Collection c2): Verifica si dos colecciones no tienen elementos en común
- frequency(Collection c, Object o): Cuenta cuántas veces aparece un objeto en una colección
- binarySearch(List list, T key): Busca un elemento en una lista ordenada
- rotate(List list, int distance): Rota los elementos de una lista

Ejemplo

```
List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);
Collections.rotate(numeros, 2);
System.out.println(numeros); // Imprime: [4, 5, 1, 2, 3]
```

Ejercicio práctico

Implementar un sistema de gestión de tareas con prioridades:

1. Crear una clase Tarea con atributos: descripción, prioridad y fecha límite
2. Usar una PriorityQueue para mantener las tareas ordenadas por prioridad
3. Implementar métodos para añadir tareas, obtener la tarea de mayor prioridad y listar todas las tareas
4. Usar un Deque para mantener un historial de tareas completadas
5. Implementar funcionalidad para deshacer la última tarea completada

Para la solución al práctico revisa en el repositorio la clase: MainTareas.java

Tarea

1. Implementa un sistema de categorías para las tareas usando un Map<String, PriorityQueue<Tarea>>

2. Añade funcionalidad para establecer recordatorios en las tareas usando un TreeSet ordenado por fecha

3. Implementa un método para obtener todas las tareas vencidas usando Streams y filtros

4. Crea un Comparator personalizado que ordene las tareas por fecha límite y luego por prioridad

5. Implementa la persistencia de tareas usando serialización de objetos

Recursos adicionales

- Java Collection Framework: docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html
- Tutorialspoint Java Collections: www.tutorialspoint.com/java/java_collections.htm

Gracias!