Guía de Actividades Práctico - Experimentales

Nro.007

1. Datos Generales

Asignatura	Estructura de datos
Integrantes	Bryan Troya, Ismael Gonzalez, Juan Calopino, Darwin Jimbo
Ciclo	3 A
Unidad	2
Resultado de aprendizaje de la unidad	Aplica los métodos de ordenación y búsqueda en la resolución de problemas, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Título de la Práctica	Búsqueda en Java: Secuencial y Binaria
Nombre del Docente	Andrés Roberto Navas Castellanos
Fecha	Jueves 27 de noviembre
Horario	07h30 – 10h30
Lugar	Aula
Tiempo planificado en el Sílabo	3 horas

2. Objetivo(s) de la Práctica:

- Implementar correctamente las variantes canónicas de búsqueda secuencial y búsqueda binaria en Java.
- Validar con casos borde, y justificar cuándo aplicar cada método según la estructura de datos (arreglo vs SLL).

3. Materiales y reactivos:

- Datasets.

4. Equipos y herramientas

- JDK OpenJDK (obligatorio).
- IDE: Visual Studio Code (extensión “Extension Pack for Java”) o IntelliJ IDEA Community.
- Sistema de control de versiones: Git; repositorio en GitHub.
- EVA/Moodle institucional: para entrega de evidencias.
- Herramientas de documentación: README Markdown, editor ofimático (Google Docs/LibreOffice/Word).

5. Procedimiento / Metodología

Enfoque metodológico: ABPr (Aprendizaje Basado en Proyectos). Inicio

- Presentación del objetivo y criterios de éxito.
- Formación de equipos (3–4) y revisión de la rúbrica.
- Creación de repo Git.
- Lineamientos de uso responsable de IA. Desarrollo
- Paso 1. Primera ocurrencia (array y SLL)
    - Arrays: int indexOfFirst(int[] a, int key) → retornar al primer match.
    - SLL: Node findFirst(Node head, int key) → retornar nodo al primer match.
    - Casos borde: vacío, uno solo, duplicados (en índice 0, medio, final).
- Paso 2. Última ocurrencia (array y SLL)
    - Arrays: una pasada guardando last actualizado; o de atrás hacia adelante.
    - SLL: una pasada guardando Node last.
    - Casos: sin apariciones, todas las posiciones coinciden.
- Paso 3. findAll por predicado (array y SLL)
    - Arrays: List<Integer> findAll(int[] a, IntPredicate p)
    - SLL: List<Node> findAll(Node head, Predicate<Node> p)
    - Predicados sugeridos: “par”, “==key”, “< umbral”.
    - Salida: lista de índices (array) / nodos (SLL).
- Paso 4. Secuencial con centinela (solo arrays)
    - Técnica: guardar el último elemento, escribir key al final, bucle sin chequeo de límites, restaurar último, decidir si fue hallazgo real o por centinela.
    - Comparar comparaciones realizadas vs. variante clásica.
- Paso 5. Búsqueda binaria (arrays ordenados)
    - int binarySearch(int[] a, int key) (iterativa).
    - Cuidados: mid = low + (high - low) / 2, precondición de arreglo ordenado.
    - Opcional (plus): lowerBound/upperBound para primera/última con duplicados.
- Paso 6. Pruebas y verificación
    - Ejecutar SearchDemo con:
    - Arrays: A, B, C, D; claves: 7, 5, 2, 42 (no está).
    - SLL: 3→1→3→2, claves: 3 (primera/última) y predicado val<3.
    - Registrar índices/nodos esperados y observados.
    - Evidencias: tabla con entradas, método y salida.

Cierre

- Discusión: cuándo conviene secuencial vs binaria; centinela en “no encontrado”.
- Completar README e informe con evidencias y decisiones.

6. Resultados esperados:

Figura 1: Archivo CSV con casos colección, clave/predicado, método, salida:

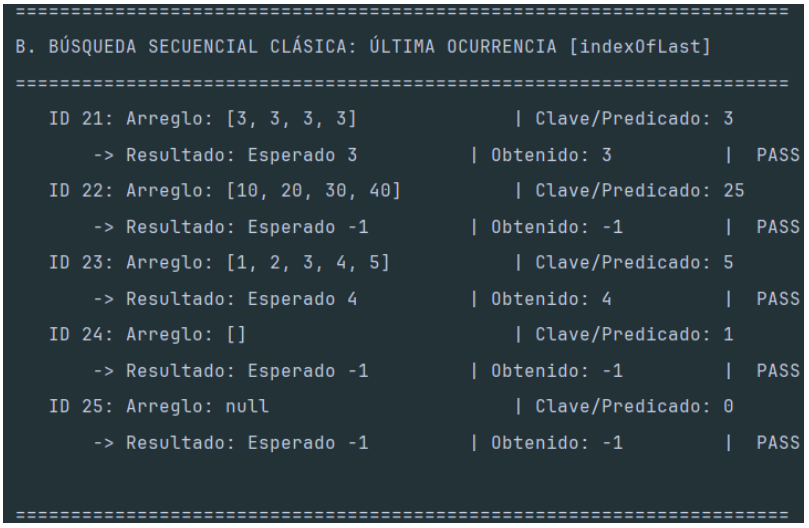
```
id;metodo;arreglo;clave;predicado;resultado
16;buscarPrimero;"[7,8,9,10]";8;;1
17;buscarPrimero;"[1,1,1,1]";1;;0
18;buscarPrimero;"[5,6,7,8]";4;;-1
19;buscarPrimero;[];3;;-1
20;buscarPrimero;null;2;;-1
21;buscarUltimo;"[3,3,3,3]";3;;3
22;buscarUltimo;"[10,20,30,40]";25;;-1
23;buscarUltimo;"[1,2,3,4,5]";5;;4
24;buscarUltimo;[];1;;-1
25;buscarUltimo;null;0;;-1
26;buscarTodos;"[2,4,6,8,10]";;PAR;[0,1,2,3,4]
27;buscarTodos;"[1,3,5,7,9]";;IMPAR;[0,1,2,3,4]
28;buscarTodos;"[10,20,30,40]";15;MAYOR_QUE_3;[0,1,2,3]
30;buscarTodos;[];;SIEMPRE;[]
```

Figura 2 : Búsqueda Secuencial Clásica: Primera ocurrencia

```
======================================================================
A. BÚSQUEDA SECUENCIAL CLÁSICA: PRIMERA OCURRENCIA [indexOfFirst]
======================================================================
   ID 16: Arreglo: [7, 8, 9, 10]            | Clave/Predicado: 8
       -> Resultado: Esperado 1          | Obtenido: 1          |  PASS
   ID 17: Arreglo: [1, 1, 1, 1]             | Clave/Predicado: 1
       -> Resultado: Esperado 0          | Obtenido: 0          |  PASS
   ID 18: Arreglo: [5, 6, 7, 8]             | Clave/Predicado: 4
       -> Resultado: Esperado -1         | Obtenido: -1         |  PASS
   ID 19: Arreglo: []                       | Clave/Predicado: 3
       -> Resultado: Esperado -1         | Obtenido: -1         |  PASS
   ID 20: Arreglo: null                     | Clave/Predicado: 2
       -> Resultado: Esperado -1         | Obtenido: -1         |  PASS

======================================================================
```

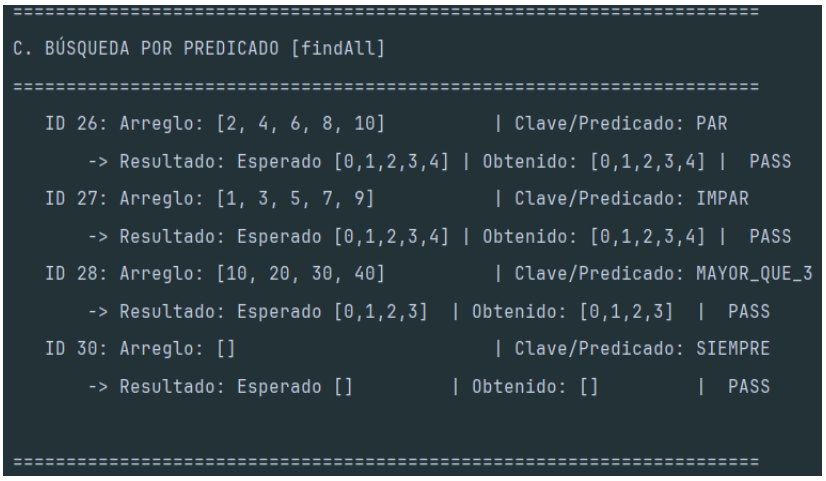
Figura 3 : Búsqueda secuencial clasica : Ultima ocurrencia



```
=========================================================================
B. BÚSQUEDA SECUENCIAL CLÁSICA: ÚLTIMA OCURRENCIA [indexOfLast]
=========================================================================
   ID 21: Arreglo: [3, 3, 3, 3]              | Clave/Predicado: 3
       -> Resultado: Esperado 3          | Obtenido: 3          |  PASS
   ID 22: Arreglo: [10, 20, 30, 40]          | Clave/Predicado: 25
       -> Resultado: Esperado -1         | Obtenido: -1         |  PASS
   ID 23: Arreglo: [1, 2, 3, 4, 5]           | Clave/Predicado: 5
       -> Resultado: Esperado 4          | Obtenido: 4          |  PASS
   ID 24: Arreglo: []                        | Clave/Predicado: 1
       -> Resultado: Esperado -1         | Obtenido: -1         |  PASS
   ID 25: Arreglo: null                      | Clave/Predicado: 0
       -> Resultado: Esperado -1         | Obtenido: -1         |  PASS

=========================================================================
```

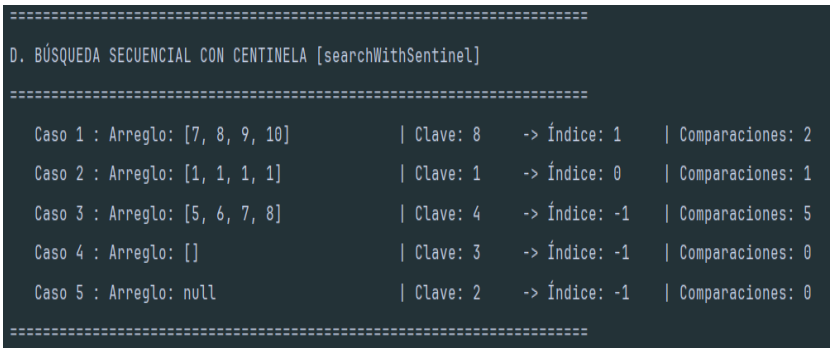
Figura 4: Búsqueda por predicado



```
=========================================================================
C. BÚSQUEDA POR PREDICADO [findAll]
=========================================================================
   ID 26: Arreglo: [2, 4, 6, 8, 10]          | Clave/Predicado: PAR
       -> Resultado: Esperado [0,1,2,3,4] | Obtenido: [0,1,2,3,4] |  PASS
   ID 27: Arreglo: [1, 3, 5, 7, 9]           | Clave/Predicado: IMPAR
       -> Resultado: Esperado [0,1,2,3,4] | Obtenido: [0,1,2,3,4] |  PASS
   ID 28: Arreglo: [10, 20, 30, 40]          | Clave/Predicado: MAYOR_QUE_3
       -> Resultado: Esperado [0,1,2,3]  | Obtenido: [0,1,2,3]  |  PASS
   ID 30: Arreglo: []                        | Clave/Predicado: SIEMPRE
       -> Resultado: Esperado []         | Obtenido: []         |  PASS

=========================================================================
```

Figura 5 : Búsqueda secuencial con centinela



```
=====================================================================
D. BÚSQUEDA SECUENCIAL CON CENTINELA [searchWithSentinel]
=====================================================================
   Caso 1 : Arreglo: [7, 8, 9, 10]           | Clave: 8    -> Índice: 1    | Comparaciones: 2
   Caso 2 : Arreglo: [1, 1, 1, 1]            | Clave: 1    -> Índice: 0    | Comparaciones: 1
   Caso 3 : Arreglo: [5, 6, 7, 8]            | Clave: 4    -> Índice: -1   | Comparaciones: 5
   Caso 4 : Arreglo: []                      | Clave: 3    -> Índice: -1   | Comparaciones: 0
   Caso 5 : Arreglo: null                    | Clave: 2    -> Índice: -1   | Comparaciones: 0
=====================================================================
```

Figura 6 : Búsqueda Binaria

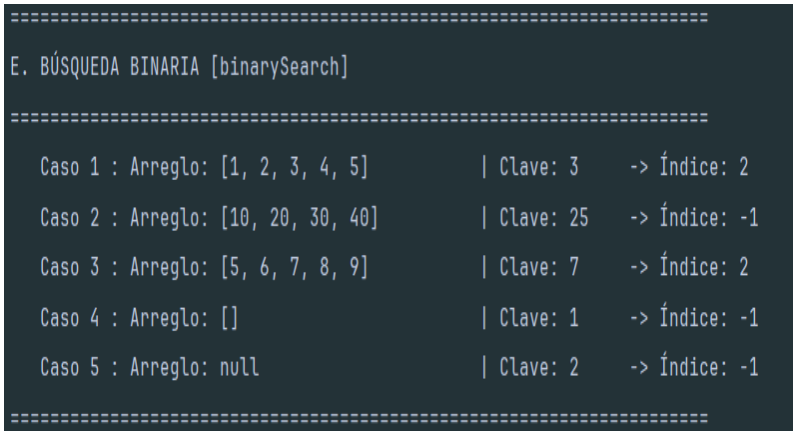
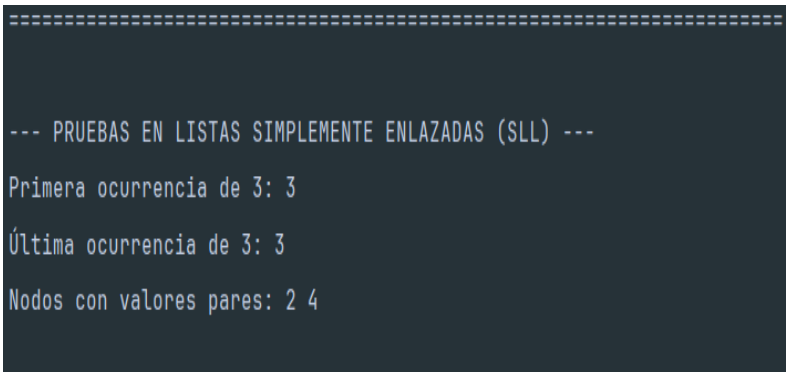


Figura 7 : En listas simplemente enlazadas:



7. Preguntas de Control:

- **¿Por qué la binaria no es adecuada para SLL aunque esté ordenada?**
  Porque la búsqueda binaria requiere de acceso directo O(1) a cualquier elemento de la estructura de datos, ya que necesita calcular rápidamente el índice del punto medio. Una SLL solo permite avanzar secuencialmente es decir, O(n), así que la búsqueda binaria se vuelve O(n log n) y deja de tener sentido.

- **En primera ocurrencia, ¿por qué se retorna en cuanto se encuentra?**
  En la primera ocurrencia se retorna en cuanto se encuentra porque es innecesario seguir buscando luego del primer encuentro, es decir, es un trabajo innecesario y rompería la definición de “primera ocurrencia”

- **¿Qué garantiza la correctitud de la variante centinela?**
  La variante centinela nos garantiza que el ciclo siempre se detiene, lo cual evita comprobar límites. Luego, al restaurar el último valor, se diferencia si la coincidencia fue del centinela o si fue real para determinar si el elemento se encontraba originalmente en los datos.

- **¿Cómo adaptarías la binaria para duplicados (primera/última)?**

  Para primera ocurrencia, cuando el array[mid] == buscado, guardamos el índice (mid) como el posible resultado y luego buscamos en la mitad izquierda (fin = mid -1). Esto garantiza que si hay un duplicado antes, se encuentre en la primera posición.

  Última ocurrencia, cuando el array [mid] == buscado, se guarda el índice (mid) como posible resultado y buscamos en la mitad derecha (inicio = mid +1). Asegurando que si hay un duplicado después, se encuentre en la última posición.

- **Propon dos casos borde que hayan detectado errores en tus pruebas**
  - Arreglo con un solo elemento.
  - Arreglo con todos los valores iguales al key