Ejemplo 1: Venta de Vehículos

Instrucciones:

Este examen práctico consta de 10 partes, cada una relacionada con una tarea específica del programa de gestión de venta de vehículos. Para cada parte, deberá crear un programa Java que implemente la funcionalidad requerida. Asegúrese de seguir las instrucciones cuidadosamente y de escribir código claro, conciso y bien documentado.

Evite utilizar programación orientada a objetos en este examen.

1. TEORÍA (1 punto): Indicar los 5 aspectos fundamentales de la programación contenciosa que permite la condensación del flujo y al presencia de los Muad'Dib's.

**3. Menú de usuario (1 punto):**

El programa debe mostrar un menú de opciones al usuario, permitiéndole interactuar con el sistema de venta de vehículos realizando diferentes operaciones.

El menú debe mostrar por consola lo siguiente:

==================================

GESTIÓN DE VENTA DE VEHÍCULOS - MENÚ DE OPCIONES

==================================

1. Agregar vehículo

2. Buscar vehículo

3. Eliminar vehículo

4. Listar vehículos

5. Ordenar vehículos alfabéticamente

6. Editar vehículo por índice

7. Salir

==================================

| public static void mostrarMenu() {  System.out.println("\n==================================");  System.out.println(" MENU DE OPCIONES");  System.out.println("==================================");  System.out.println("1. Agregar auto");  System.out.println("2. Buscar auto");  System.out.println("3. Eliminar auto");  System.out.println("4. Listar autos");  System.out.println("5. Ordenar autos alfabéticamente");  System.out.println("6. Editar auto indicando índice");  System.out.println("7. Salir");  System.out.println("==================================\n");  System.out.println("Por favor, ingrese el número de la opción que desea realizar:");  } |
| --- |

**4. Navegación (1 punto):**

El programa debe permitir al usuario realizar las operaciones del menú siempre que no seleccione la opción de salir. Después de ejecutar una opción, el menú debe volver a mostrarse.

| do {   mostrarMenu();  opcion = scanner.nextInt();  scanner.nextLine();   switch (opcion) {  case 1:  System.out.print("Ingrese el nombre del auto: ");  String nuevoAuto = scanner.nextLine();  agregarAuto(autos, nuevoAuto);  break;  case 2:  System.out.print("Ingrese el nombre del auto a buscar: ");  String autoABuscar = scanner.nextLine();  buscarAuto(autos, autoABuscar);  break;  case 3:  System.out.print("Ingrese el nombre del auto a eliminar: ");  String autoAEliminar = scanner.nextLine();  eliminarAuto(autos, autoAEliminar);  break;  case 4:  System.out.println(listarAutos(autos));  break;  case 5:  ordenarAutos(autos);  break;  case 6:  System.out.print("Ingrese el índice del auto a editar: ");  int indice = scanner.nextInt();  scanner.nextLine();  System.out.print("Ingrese el nuevo nombre del auto: ");  String nuevoNombre = scanner.nextLine();  editarAuto(autos, indice, nuevoNombre);  break;  case 7:  System.out.println("Saliendo del programa.");  break;  default:  System.out.println("Opción no válida.\n");  break;  }  } while (opcion != 7); |
| --- |

**5. Agregar vehículo (1 punto):**

Declarar la estructura de datos ArrayList<String> para almacenar los vehículos en el sistema.

Implementar el método agregarVehiculo que permita agregar vehículos, recibiendo como parámetros de entrada el ArrayList y el nombre del vehículo.

| public static void agregarAuto(ArrayList<String> autos, String nombreauto) {  autos.add(nombreauto);  System.out.println("auto agregado correctamente.");  } |
| --- |

**6. Buscar vehículo (1 punto):**

Implementar el método buscarVehiculo que permita buscar un vehículo por su nombre en el ArrayList y mostrar su posición en la lista, o indicar que no se encuentra con -1.

| public static void buscarAuto(ArrayList<String> autos, String nombreauto) {  int indice = autos.indexOf(nombreauto);  if (indice != -1) {  System.out.println("auto encontrado en la posición: " + indice);  } else {  System.out.println("auto no encontrado.");  }  } |
| --- |

**7. Eliminar vehículo (1 punto):**

Implementar el método eliminarVehiculo que permita eliminar un vehículo por su nombre del ArrayList.

| public static void eliminarAuto(ArrayList<String> autos, String nombreauto) {  if (autos.remove(nombreauto)) {  System.out.println("auto eliminado correctamente.");  } else {  System.out.println("auto no encontrado.");  }  } |
| --- |

**8. Listar vehículos (1 punto):**

Implementar el método listarVehiculos que muestre todos los vehículos presentes en el ArrayList.

| public static String listarAutos(ArrayList<String> autos) {  System.out.println("Listado de autos:");  String listaautos = "";  for (String auto : autos) {  listaautos += auto + "\n";  }  return listaautos;  } |
| --- |

**9. Ordenar vehículos alfabéticamente (1 punto):**

Implementar el método ordenarVehiculos que ordene los vehículos en el ArrayList en orden alfabético recibiendo el array a ordenar.

| public static void ordenarAutos(ArrayList<String> autos) {  Collections.sort(autos);  System.out.println("autos ordenados alfabéticamente.");  } |
| --- |

**10. Editar un vehículo por índice (1 punto):**

Implementar el método editarVehiculo que permita editar el nombre de un vehículo por su índice en el ArrayList.

| public static void editarAuto(ArrayList<String> autos, int indice, String nuevoNombre) {  if (indice >= 0 && indice < autos.size()) {  autos.set(indice, nuevoNombre);  System.out.println("auto editado correctamente.");  } else {  System.out.println("Índice no válido.");  }  } |
| --- |

Ejemplo 2: La Biblioteca

## **INSTRUCCIONES:**

1. Implementar un **programa en Java que simula el funcionamiento de una biblioteca**, utilizando la estructura de datos **ArrayList**.
2. El programa debe estar contenido en un único archivo llamado "**Biblioteca.java**".
3. No se permite el uso de librerías externas.
4. Los métodos utilizados deben ser estáticos.

## **CUESTIONARIO:**

1) TEORÍA *(1 p.)*

2) TEORÍA *(1 p.)*

3) **Menú de usuario** *(1 p.)*: El programa debe mostrar un menú de opciones al usuario, permitiéndole interactuar con la biblioteca realizando diferentes operaciones.

El menú debe mostrar por consola lo siguiente:

==================================

BIBLIOTECA - MENÚ DE OPCIONES

==================================

1. Agregar libro

2. Buscar libro

3. Eliminar libro

4. Listar libros

5. Ordenar libros alfabéticamente

6. Editar libro indicando índice

7. Salir

==================================

4) **Navegación** *(1 p.)***:** El programa debe permitir al usuario realizar las operaciones del menú siempre que no seleccione la opción de salir. Después de ejecutar una opción, el menú debe volver a mostrarse.

5) **Agregar Libros** *(1 p.)***:** Declarar la estructura de datos ArrayList<String> para almacenar los libros de la biblioteca.

Implementar el método agregarLibro que permita agregar libros, recibiendo como parámetros de entrada el ArrayList y el nombre del libro.

6) **Buscar Libro** *(1 p.)*: Implementar el método buscarLibro que permita buscar un libro por su nombre en el ArrayList y mostrar su posición en la lista, o indicar que no se encuentra.

7) **Eliminar libro** *(1 p.)*: Implementar el método **eliminarLibro** que permite eliminar un libro por su nombre del ArrayList.

8) **Listar libro** *(1 p.)*: Implementar el método **listarLibros** que muestre todos los libros presentes en el ArrayList.

9) **Ordenar libros alfabéticamente** *(1 p.)*: Implementar el método **ordenarLibros** que ordene los libros en el ArrayList en orden alfabético.

10) **Editar un libro indicando su índice** *(1 p.)*: Implementar el método editarLibro que permita editar el nombre de un libro indicando su índice en el ArrayList.