Examen_0 Gestión de Asignaturas en una Academia igformació iG Estás desarrollando una aplicación para gestionar las asignaturas en distintos cursos de una academia. Para ello, vas a crear una pequeña aplicación en Java usando ArrayList Crea una clase principal (Main) que muestre un menú interactivo en consola con las siguientes opciones:

1. Añadir asignatura
2. Buscar asignatura por nombre
3. Modificar profesor titular
4. Eliminar asignatura
5. Mostrar asignaturas del curso
6. Copiar asignaturas a otro curso 7. Salir Crea una clase Curso que contenga: • Un atributo ArrayList para almacenar las asignaturas en ese curso. Crea una clase llamada Asignatura que contenga nombre,horas semanales, profesor, presencial (bool). • Un constructor para inicializar todos los atributos. • Un método toString() que devuelve una representación de la asignatura en el siguiente formato: "Nombre - Profesor - Horas: X - Presencial: sí/no" Incluye métodos para realizar las sigulentes operaciones:
7. Añadir una asignatura.
8. Buscar una asignatura por su nombre.
9. Modificar el email de una asignatura buscando su nombre.
10. Eliminar a una asignatura por su nombre.
11. Mostrar todas las asignaturas matriculadas, una por línea, en el formato proporcionado por toString().
12. Copiar todas las asignaturas de un curso a otro.

IG Igformación Ahora,realiza una segunda parte del ejercicio usando una LinkedList para simular una lista de asignaturas del curso (por ejemplo: "Java", "POO", "Estructuras de datos", etc.).

1. Crear la LinkedList e insertar varios elementos al inicio y al final de la lista.
2. Mostrar los elementos de la lista en orden (usando Iterator).
3. Mostrar los elementos en orden inverso (usando ListIterator).
4. Modificar un elemento de la lista (por ejemplo, cambiar "Java" por "Java Avanzado").
5. Eliminar un elemento de la lista por su contenido. Puedes crear un nuevo menú específico para esta parte o integrarlo en el anterior,según prefieras. NOTA: Se valorará el uso adecuado de modularización (métodos separados para cada funcionalidad). Asegúrate de validar entradas donde sea necesario. El uso correcto de ArrayList, LinkedList, objetos y estructuras de control es fundamental.

Examen1 – Opción A: Gestor de Biblioteca

I. Colecciones: Set, Map y TreeSet

1. Crea una clase Libro con atributos: ISBN (String), título, autor, disponible (boolean).
2. En la clase principal (Main), presenta un menú consola con opciones:
    a. Añadir libro (usa un **HashSet** para evitar duplicados por ISBN).
    b. Buscar libro por ISBN (consulta en un **HashMap<String,Libro>**).
    c. Modificar estado de disponibilidad (marcar prestado/devolución).
    d. Eliminar libro (de ambos, Set y Map).
    e. Mostrar todos los libros ordenados por título (utiliza un **TreeSet** con Comparator<Libro>).
    f. Copiar la colección de libros disponibles a un nuevo TreeSet.
    g. Salir.

II. Operaciones con ficheros

7. Escribir en un fichero de texto (biblioteca.txt) la lista de libros (uno por línea, en formato "ISBN;Título;Autor;Disponible").
8. Mostrar por consola el contenido completo de biblioteca.txt.
9. Leer biblioteca.txt y reconstruir la colección de Libro en memoria.
10. Modificar en el fichero la disponibilidad de un libro buscando por ISBN (por ejemplo, cambiar "false" a "true").

III. Genéricos y lambdas

11. Crea una clase genérica Repositorio<T> con métodos: añadir(T elemento), eliminar(T elemento), List<T> listar().
12. Instancia un Repositorio<Libro> y, usando **Streams** + **lambdas**, muestra por consola los títulos de los libros cuya disponibilidad sea true.
13. Usando un Function<Libro,String> y Consumer<String>, imprime autor y título en formato "Autor: **, Libro: **".

IV. Expresiones regulares

14. Valida el formato del ISBN: debe coincidir con el patrón \d{3}-\d{10}.
15. Lee una línea de descripción de libro y reemplaza con regex todas las ocurrencias de más de una palabra en mayúsculas por la misma en minúsculas.

Examen_2– Opción B: Gestor de Empleados

I. Colecciones: Set, Map y TreeSet

1. Crea una clase Empleado con: id (int), nombre, departamento, salario (double).
2. Menú consola:
    a. Añadir empleado (usa **HashSet** para IDs únicos).
    b. Buscar empleado por ID (usa **HashMap<Integer,Empleado>**).
    c. Actualizar salario de un empleado.
    d. Eliminar empleado.
    e. Mostrar todos los empleados ordenados por salario (con **TreeSet** y Comparator<Empleado>).
    f. Copiar empleados de un departamento a un nuevo TreeSet.
    g. Salir.

II. Operaciones con ficheros

7. Generar un fichero empleados.csv con “id,nombre,departamento,salario”.
8. Mostrar su contenido en pantalla.
9. Leer empleados.csv e insertar cada Empleado en las colecciones de memoria.
10. Modificar en el CSV el departamento de un empleado dado su ID.

III. Genéricos y lambdas

11. Define un genérico Almacen<T> con void guardar(T obj), T obtener(int índice), List<T> todos().
12. Con **Stream API** + lambdas filtra empleados con salario > X y ordénalos por nombre.
13. Usa un Predicate<Empleado> para comprobar si un empleado pertenece a “RRHH” y un Consumer<Empleado> para imprimir su ficha completa.

IV. Expresiones regulares

14. Comprueba que el nombre contenga solo letras y espacios (^[A-Za-z]+$).
15. Dada una línea de texto con código de departamento (ej. “DEP-1234”), extrae numéricamente los dígitos con regex y muéstralos.

Notas comunes a ambos exámenes

- Organiza cada funcionalidad en métodos independientes.
- Valida entradas (p.ej. que el ID o ISBN no estén vacíos o mal formateados).
- Usa Iterator o ListIterator donde tenga sentido al mostrar o recorrer colecciones.
- Asegúrate de manejar excepciones de I/O y posibles errores de formato en regex.

Examen3 – Opción C: Gestor de Inventario

I. Colecciones: Set, Map y TreeSet

1. Define la clase Producto con campos:
    a. id (String)
    b. nombre (String)
    c. precio (double)
    d. stock (int)
2. En Main, muestra un menú de consola con estas opciones:
    a. **Añadir producto** (usa un **HashSet** para evitar duplicados por id).
    b. **Buscar producto** por id (usa un **HashMap<String,Producto>**).
    c. **Actualizar stock** de un producto.
    d. **Eliminar producto** (de Set y Map).
    e. **Mostrar todos los productos** ordenados por nombre (usa un **TreeSet** con Comparator<Producto>).
    f. **Copiar** todos los productos con stock > 0 a un nuevo TreeSet.
    g. **Calcular valor total del inventario**: recorre la colección y suma precio * stock.
    h. Salir.

Métodos sugeridos:

```
boolean agregarProducto(Producto p);
Producto buscarPorId(String id);
boolean actualizarStock(String id, int nuevoStock);
boolean eliminarProducto(String id);
Set<Producto> listarOrdenadosPorNombre();
double calcularValorInventario();
```

II. Operaciones con ficheros

7. **Escribir** en inventario.txt la lista de productos (formato CSV: id;nombre;precio;stock).
8. **Mostrar** por consola el contenido de inventario.txt.
9. **Leer** inventario.txt y reconstruir la colección de Producto.
10. **Modificar** en el fichero el stock de un producto dado su id.

```
void escribirFichero(Path ruta);
List<String> leerFichero(Path ruta);
void actualizarStockEnFichero(Path ruta, String id, int stock);
```

III. Genéricos y lambdas

1. Crea Repositorio<T> con:

```
void guardar(T elemento);
void eliminar(T elemento);
```

List<T> listar();

2. Instancia Repositorio<Producto>, y con **Stream + lambda**:
    a. Filtra los productos con stock < 5 (stock bajo) y muéstralos.
    b. Ordena los productos por precio de menor a mayor.
3. Usa un BiFunction<Double, Double, Double> para aplicar un **descuento** al precio (p.ej. nuevoPrecio = precio × (1 – descuento)), y un Consumer<Producto> para imprimir ficha.

IV. Expresiones regulares

14. **Valida** que el id cumpla el patrón: prefijo “PRD-” seguido de 4 dígitos (PRD-\d{4}).
15. Dada una línea con formato "PRD-1234: Tablet Pro (299.99€)", **extrae** con regex el código (PRD-1234), el nombre y el precio (sólo la parte numérica).

Notas para el alumno

- Separa cada funcionalidad en métodos bien nombrados.
- Maneja excepciones de I/O y NumberFormatException.
- Valida las entradas (p.ej. precio ≥ 0, stock ≥ 0).
- Emplea Comparator, Iterator y Streams donde corresponda para reforzar todos los conceptos.

Examen 4

Una estación meteorológica registra lecturas diarias de temperatura y humedad. Se necesita un sistema para almacenar las lecturas, consultarlas y exportar datos a un archivo.

Requisitos:Clases Base

- Clase Lectura: LocalDate fecha, double temperatura, double humedad
- Métodos: toString(), equals() y hashCode() por fecha
- Clase anidada estática TipoDato (TEMPERATURA, HUMEDAD)
- Interfaz funcional Conversor con método double convertir(double valor)
- Implementa una lambda para convertir °C a °F

Clase Estacion

Atributos:

- Set<Lectura> lecturas
- Map<LocalDate, Lectura>
- Métodos:
- agregarLectura(Lectura)
- buscarPorFecha(LocalDate)
- mostrarLecturasOrdenadas() (por fecha)

- exportarFichero(String nombreFichero) usando FileWriter

Main

- Menú para:
- Añadir lectura
- Buscar por fecha
- Mostrar ordenado
- Convertir temperatura a °F con lambda
- Guardar en archivo

Examen 5 – Registro de Mascotas en una Clínica Veterinaria

Temas: Clases anidadas, Set, Predicate, Consumer, lambda, var, genéricosContexto:

Una clínica registra mascotas por tipo (perro, gato, etc.), peso, y fecha de ingreso. Se quiere

gestionar los pacientes, aplicar tratamientos y detectar casos críticos.

Requisitos:

Clases Base

- Clase Mascota: String nombre, String tipo, double peso, LocalDate ingresoMétodos: toString(), equals() y hashCode() por nombre
- Clase anidada estática TipoMascota con constantes (PERRO, GATO, OTRO)
- Interfaz funcional Tratamiento con método void aplicar(Mascota m)
- Crear varias lambdas para representar tratamientos diferentes
- Clase Clinica<T extends Mascota>

Atributo: Set<T> pacientes

Métodos:

- agregarMascota(T mascota)
- filtrarPorPeso(Predicate<Mascota>)
- aplicarTratamientos(Consumer<Mascota>)
- mostrarTodos()

Main

- Menú:
- Añadir mascota
- Aplicar tratamiento (lambda)
- Mostrar mascotas con sobrepeso (Predicate)
- Mostrar todas

Examen 6 – Control de Cursos Universitarios

Temas: Clases anidadas, Map, Set, colecciones ordenadas, Function, Supplier, final, var,

ficheros, interfacesContexto:

Un sistema gestiona cursos, profesores y estudiantes. Se deben registrar inscripciones, calcular notas finales, y exportar listados.

Requisitos:

Clases Base

- Clase Curso: String nombre, Map<String, Double> notas (clave = estudiante)
- Método para añadir nota, calcular promedio
- Clase Profesor: String nombre, Set<Curso> cursosAsignados
- Método para añadir curso
- Interfaz funcional CalculadoraNotaFinal con método double calcular(Map<String, Double> notas)
- Implementa una lambda para obtener la media
- Clase UniversidadAtributos:
- Map<String, Profesor> profesores (clave = nombre)
- Set<Curso> todosLosCursos

Métodos:

- agregarProfesor(Profesor)
- mostrarProfesores()
- calcularPromedios(Function<Map<String, Double>, Double>)
- guardarEnArchivo(String archivo)

Main

Menú:

- Añadir profesor y curso
- Asignar notas
- Calcular promedio con lambdaExportar listado