Diseño de clases

Ejercicio 1: Calculadora

Diseña una clase en Java llamada Calculadora que permita realizar operaciones básicas entre dos números. Esta clase debe tener métodos públicos que permitan sumar, restar, multiplicar y dividir dos números reales. Asegúrate de que el método de división verifique que el divisor no sea cero antes de realizar la operación, para evitar errores. Puedes hacer que todos los métodos devuelvan el resultado como un número de tipo double.

Ejercicio 2: Perro

Crea una clase llamada Perro que represente a una mascota. Esta clase debe tener tres atributos: nombre, raza y edad. Además, debe incluir un método llamado ladrar que imprima en consola un mensaje como "¡Guau guau!", un método cumplirAnios que aumente la edad del perro en uno, y un método mostrarInformacion que muestre por consola todos los datos del perro. Puedes inicializar los atributos a través de un constructor.

Ejercicio 3: Película

Implementa una clase llamada Pelicula que contenga información sobre una película. Esta clase debe tener como atributos el título de la película, el nombre del director y la duración en minutos. Debe incluir un constructor para establecer estos valores al crear un objeto, y un método llamado esLarga que devuelva true si la película dura más de 120 minutos, y false en caso contrario.

Ejercicio 4: Biblioteca y Libros

Diseña una clase llamada Libro que contenga atributos como el título, el autor y el número de páginas. Luego, crea una clase llamada Biblioteca, la cual debe contener una lista de libros. La clase Biblioteca debe permitir agregar libros a la colección, eliminar un libro por su título y mostrar todos los libros disponibles. Además, implementa un método que devuelva el libro con más páginas.

Ejercicio 5: Cuenta bancaria

Crea una clase llamada CuentaBancaria con los atributos titular, numeroCuenta y saldo. Implementa métodos para ingresar dinero, retirar dinero (si hay saldo suficiente), y mostrar el saldo actual. Luego, añade una clase llamada Banco que gestione múltiples cuentas. Esta clase debe poder agregar nuevas cuentas, buscar una cuenta por su número y mostrar un resumen general de todas las cuentas (incluyendo saldo y titular).

Ejercicio 6: Sistema de gestión de estudiantes

Implementa una clase llamada Estudiante con atributos como nombre, ID y una lista de calificaciones. Añade métodos para agregar una nueva calificación, calcular el promedio y determinar si el estudiante está aprobado (por ejemplo, si su promedio es igual o mayor a 5). Luego, crea una clase Clase que contenga un conjunto de estudiantes y permita añadir estudiantes, buscar a uno por su ID y mostrar a todos los estudiantes aprobados.

Ejercicio 7: Biblioteca y Libros Plus

Diseña un sistema completo de biblioteca en el que la clase Libro incluya ISBN, título, autor, número de páginas y un indicador de disponibilidad; crea además la clase Usuario con un identificador único, nombre y un nivel de membresía que limite la cantidad simultánea de préstamos; implementa la clase Prestamo, que registre libro, usuario, fecha de inicio, fecha límite y fecha real de devolución, calculando automáticamente días de retraso y multa; la clase Biblioteca debe mantener colecciones de libros, usuarios y préstamos, permitir alta y baja de usuarios, préstamo, devolución, renovación, reservas en cola, generación de informes ordenados (por título, autor o número de páginas usando Comparable y Comparator), persistir todos los datos en formato JSON con recuperación al iniciar el programa, y lanzar excepciones personalizadas cuando se intente prestar un libro no disponible, superar el cupo de préstamos o devolver con retraso.

Opcional: añade una interfaz Identificable que contenga el método getId() y haz que tanto Libro como Usuario la implementen. Esto permitirá aplicar técnicas de búsqueda genéricas con colecciones.

Ejercicio 8: Cuenta bancaria Plus

Crea una jerarquía de clases para modelar un sistema bancario. Comienza con una clase abstracta llamada CuentaBancaria, que contenga los atributos comunes: titular (nombre del cliente), número de cuenta IBAN (único) y saldo. Define en esta clase los métodos abstractos aplicarComisiones() y mostrarResumen(), y añade métodos concretos como ingresar(double cantidad) y retirar(double cantidad), asegurando que no se pueda retirar más de lo que hay disponible.

Extiende esta clase con CuentaCorriente, que implemente una comisión fija mensual, y CuentaAhorro, que tenga una tasa de interés anual y calcule el interés mensual acumulado. Implementa ambos métodos abstractos en cada subclase de forma adecuada.

Crea una clase Banco que utilice una colección Map<String, CuentaBancaria> para almacenar todas las cuentas, usando el IBAN como clave. Esta clase debe permitir crear nuevas cuentas (verificando que no exista otra con el mismo IBAN), cerrar cuentas, buscar cuentas por su IBAN y generar un listado con todos los resúmenes de las cuentas.

Además, crea una interfaz Transferible que defina el método transferir(CuentaBancaria destino, double cantidad), y haz que CuentaCorriente y CuentaAhorro la implementen. El método

transferir debe comprobar que el saldo sea suficiente y descontar la cantidad de una cuenta, añadiéndola a la otra.

Finalmente, implementa una clase Movimiento con información sobre las operaciones realizadas (tipo, cantidad, fecha, cuenta origen y cuenta destino si aplica), y haz que cada cuenta mantenga una lista de sus movimientos. Incluye un método en la clase Banco que muestre todos los movimientos realizados por una cuenta específica.

Ejercicio 9: Sistema de gestión de estudiantes
Diseña un sistema académico que permita gestionar estudiantes, profesores y asignaturas. Comienza creando la clase Estudiante, que debe contener el nombre, un identificador único y un mapa que relacione el nombre de cada asignatura con una lista de calificaciones (Map<String, List<Double>>). Añade un método que calcule la nota media global del estudiante, y otro que calcule su GPA ponderado, asumiendo que cada asignatura tiene un peso que se obtiene del sistema.

Define la interfaz Evaluable con un método calcularNotaFinal(). Implementa esta interfaz en la clase Asignatura, que debe tener un identificador, nombre, una lista de evaluaciones (cada una con nota y porcentaje) y un método para calcular la nota final del alumno a partir de sus calificaciones. Crea una clase Evaluacion que contenga la nota y el porcentaje que representa dentro de la asignatura.

Crea la clase Profesor, que tendrá nombre, identificador y una lista de asignaturas que imparte. Debe contener métodos para añadir evaluaciones a un estudiante concreto en una asignatura, y para listar todos sus alumnos.

Por último, implementa la clase Clase, que agrupe a varios estudiantes, profesores y asignaturas. Debe permitir matricular estudiantes en asignaturas, asignar profesores a asignaturas, y registrar evaluaciones. Añade métodos para obtener un ranking de los estudiantes según su media global, buscar estudiantes por ID y listar las notas de una asignatura.

Define excepciones personalizadas como MatriculaDuplicadaException, NotaFueraDeRangoException o EstudianteNoEncontradoException. Usa colecciones adecuadas (Map, Set, List) y aplica principios de encapsulamiento, herencia e interfaces donde tenga sentido.

Colecciones

Ejercicio 1: Lista de nombres

Crea un programa que permita almacenar nombres en una lista (ArrayList). El usuario podrá añadir nombres, eliminar uno por su contenido (si existe), mostrar la lista completa y buscar si un nombre está presente. Los métodos deben estar organizados en una clase llamada GestorDeNombres.

Ejercicio 2: Contador de palabras
Diseña un programa que lea una serie de palabras desde consola o desde un arreglo predefinido, y cuente cuántas veces aparece cada palabra. Usa un HashMap<String, Integer> para almacenar cada palabra junto con su número de repeticiones. El resultado debe imprimirse mostrando cada palabra y su frecuencia.

Ejercicio 3: Ordenar una lista de números
Crea un programa que pida al usuario una serie de números enteros, los almacene en una lista (ArrayList<Integer>) y luego los ordene de menor a mayor. Muestra la lista original y la lista ordenada. Implementa esto en una clase llamada OrdenadorDeNumeros.

Ejercicio 4: Gestión de productos en stock
Crea una clase Producto con los atributos codigo, nombre, precio y cantidadEnStock. Implementa los métodos equals y hashCode para que dos productos se consideren iguales si tienen el mismo código. Luego, crea una clase Inventario que use un HashSet para almacenar productos sin duplicados. La clase debe permitir añadir nuevos productos, actualizar el stock de uno existente, eliminar productos por su código, y mostrar todos los productos ordenados por nombre o precio utilizando Comparator. También debe incluir un método que devuelva el producto con mayor cantidad en stock.

Ejercicio 5: Registro de votos
Diseña un sistema de votación simple. Crea una clase Eleccion que contenga un Map<String, Integer> donde la clave es el nombre del candidato y el valor es la cantidad de votos recibidos. El sistema debe permitir registrar un voto para un candidato (añadiéndolo si aún no existe), obtener el número total de votos, mostrar los resultados ordenados de mayor a menor cantidad de votos, y determinar el ganador.

Incluye también un método para eliminar un candidato del registro, por si se retira de la elección.

Ejercicio 6: Diccionario multilingüe
Implementa una clase Diccionario que permita almacenar traducciones de palabras en varios idiomas. Utiliza un Map<String, Map<String, String>>, donde la primera clave es la palabra base (por ejemplo, en español), y el valor es otro mapa que relaciona un idioma con su traducción (por ejemplo, "en" → "apple", "fr" → "pomme"). La clase debe permitir añadir una nueva palabra con sus traducciones, obtener la traducción de una palabra en un idioma específico, actualizar una traducción existente, eliminar una palabra completamente y mostrar todas las traducciones disponibles ordenadas alfabéticamente por la palabra base.

Ejercicio 7: Gestión de biblioteca técnica
Crea una clase LibroTecnico con atributos como ISBN, título, autor, temas (una lista de Strings) y número de ediciones publicadas. Implementa Comparable para ordenar por número de ediciones y crea al menos dos Comparator distintos: uno para ordenar por título y otro por autor. Diseña la clase CatalogoTecnico, que contenga una colección de libros (puedes usar un TreeSet o ArrayList) y permita buscar libros por tema (debe devolver todos los libros que contengan el tema buscado), agrupar los libros por autor usando Map<String, List<LibroTecnico>>, y generar un informe de los autores ordenados alfabéticamente con la cantidad total de libros que tienen en el catálogo. Agrega también un método que devuelva los 3 libros con más ediciones.

Ejercicio 8: Sistema de análisis de encuestas
Implementa una clase Encuesta que contenga un identificador, el texto de la pregunta y una lista de respuestas recibidas (como cadenas de texto). Crea una clase AnalizadorEncuestas que permita gestionar múltiples

encuestas en una colección Map<Integer, Encuesta>. Debe ofrecer métodos para registrar una nueva respuesta en una encuesta específica, contar cuántas veces aparece una determinada respuesta en una encuesta, y calcular la frecuencia de cada respuesta como porcentaje, almacenando estos datos en un nuevo Map<String, Double> donde la clave es la respuesta y el valor es su porcentaje sobre el total.

Añade también un método que devuelva las tres respuestas más comunes, ordenadas de mayor a menor cantidad de apariciones. Para ello, puedes recorrer la lista de respuestas, construir un nuevo Map<String, Integer> con los conteos manualmente (usando un bucle), y luego ordenar este mapa en función de los valores. Asegúrate de que, si una encuesta no tiene respuestas aún, los métodos no fallen, y lanza una excepción personalizada si se intenta acceder a una encuesta que no existe.

Ejercicio 9: Red de ciudades y rutas

Crea una clase Ciudad con nombre, código postal y una lista de ciudades conectadas (simulando rutas directas entre ellas). Implementa una clase RedDeCiudades que almacene todas las ciudades en un Map<String, Ciudad> y permita añadir rutas entre ellas. Implementa un método que, dada una ciudad de origen, devuelva todas las ciudades accesibles desde ella recorriendo conexiones directas (puedes usar una búsqueda en profundidad usando un Set para evitar repeticiones). Añade otro método que devuelva las ciudades ordenadas por el número de conexiones descendente. Finalmente, implementa un método para buscar si existe un camino entre dos ciudades, sin necesidad de mostrar la ruta, solo indicando si están conectadas directa o indirectamente.

Bases de datos con ventanas

Ejercicio 1: Agenda de contactos
Crea una aplicación con una ventana básica (usando JFrame) que permita al usuario gestionar una agenda de contactos. La interfaz debe tener campos para nombre, teléfono y correo electrónico, así como botones para añadir, eliminar y listar contactos. Los datos deben almacenarse en una base de datos (por ejemplo, SQLite) en una tabla llamada contactos. Al iniciar la aplicación, debe mostrarse una lista con los contactos ya guardados.

Ejercicio 2: Gestión de productos
Diseña una aplicación de escritorio con una interfaz gráfica sencilla que permita gestionar productos de una tienda. La ventana debe tener campos para introducir nombre, precio y stock de un producto. Añade botones para guardar un nuevo producto, eliminarlo por su ID y mostrar la lista completa. Los productos deben guardarse en una base de datos con una tabla productos. Al pulsar en “Mostrar todos”, los datos deben cargarse en una tabla (JTable).

Ejercicio 3: Registro de estudiantes
Implementa una aplicación gráfica en Java que permita registrar estudiantes con nombre, curso y nota media. Los datos deben guardarse en una base de datos en la tabla estudiantes. La interfaz debe permitir insertar nuevos registros, buscar estudiantes por nombre (mostrando resultados en una lista o tabla), y eliminar estudiantes seleccionados. Asegúrate de validar que la nota sea un número entre 0 y 10 antes de guardar.