1. Sistema de Gestión de Vehículos y su Alquiler

Diseña un sistema para gestionar vehículos.

- Clase Abstracta: Vehiculo (con atributos como matricula, modelo, costoBaseDia) y un método abstracto calcularCostoAlquiler(int dias) para determinar el costo total.
- Clases Concretas: Coche (con atributo numPuertas) y Camioneta (con atributo capacidadCarga), ambas heredando de Vehiculo y sobreescribiendo el método calcularCostoAlquiler para aplicar un recargo específico (e.g., el coche tiene un recargo fijo, la camioneta uno por cada 100 kg de carga).
- Interfaz: Alquilable con un método mostrarDetalleAlquiler(). Ambas clases concretas deben implementar esta interfaz.
- Estructura: La clase principal (Main) debe usar un ArrayList<Vehiculo> para almacenar los vehículos disponibles.
- Menú: Implementa un menú que permita: 1) Agregar un nuevo vehículo (usando un constructor), 2) Mostrar la lista de todos los vehículos (demostrando polimorfismo al iterar el ArrayList), 3) Calcular el costo de alquiler para un vehículo específico, 4) Salir.

2. Sistema de Gestión de Personal en una Empresa

Crea un sistema para administrar diferentes tipos de empleados.

- Clase Abstracta: Empleado (con atributos como nombre, id, salarioBase) y un método abstracto calcularSalarioNeto(). Incluye un constructor.
- Clases Concretas: Desarrollador y Gerente. Ambas heredan de Empleado.
- Desarrollador tiene el atributo lenguajePrincipal y sobreescribe calcularSalarioNeto para aplicar una bonificación fija.
- Gerente tiene el atributo departamento y sobreescribe calcularSalarioNeto para aplicar una bonificación porcentual.
- Interfaz: Reporteador con un método generarReporte(String tipoReporte). Solo la clase Gerente debe implementar esta interfaz.
- Estructura: La clase principal debe mantener una lista de empleados usando un ArrayList<Empleado>. Utiliza estructuras de control (if/else o switch) para las bonificaciones.
- Menú: El menú principal debe tener opciones para: 1) Contratar un nuevo empleado (agregándolo al ArrayList), 2) Mostrar el salario neto de todos los empleados (aplicando polimorfismo), 3) Ejecutar un método de Reporteador para un gerente específico (demostrando el uso de la interfaz y un cast de objeto), 4) Salir.

3. Simulación de Biblioteca Digital

Desarrolla una simulación para gestionar ítems de una biblioteca.

- Interfaz: Prestable con métodos prestar() y devolver().

- Clase Abstracta: ItemBiblioteca (con atributos como titulo, autor, codigo) y un método abstracto obtenerInformacion().
- Clases Concretas: Libro y Revista. Ambas heredan de ItemBiblioteca y tienen un constructor.
- Libro (con atributo ISBN) debe implementar la interfaz Prestable.
- Revista (con atributo numeroEdicion) debe sobreescribir obtenerInformacion para incluir el número de edición.
- Estructura: Utiliza un ArrayList<ItemBiblioteca> para la colección de la biblioteca. Las operaciones de menú deben usar estructuras de control para buscar ítems por código.
- Menú: Ofrece un menú para: 1) Añadir un Libro o Revista (creando un objeto con un constructor), 2) Mostrar todos los ítems (ejemplo de polimorfismo), 3) Prestar un ítem (solo si es un Libro, demostrando el uso de la interfaz), 4) Salir.

4. Sistema de Cuentas Bancarias

Implementa un sistema básico de gestión de cuentas.

- Clase Abstracta: CuentaBancaria (con atributos numeroCuenta, saldo, titular) y métodos abstractos depositar(double monto) y retirar(double monto).
- Clases Concretas: CuentaAhorro y CuentaCorriente. Ambas heredan de CuentaBancaria y tienen un constructor.
- CuentaAhorro (con atributo tasaInteres) sobreescribe retirar para aplicar un límite.
- CuentaCorriente sobreescribe retirar para permitir un sobregiro limitado.
- Interfaz: Transaccionable con un método aplicarCargoMensual().
- CuentaAhorro implementa Transaccionable para añadir interés.
- CuentaCorriente implementa Transaccionable para cobrar una comisión.
- Estructura: Almacena todas las cuentas en un ArrayList<CuentaBancaria>.
- Menú: El menú debe permitir: 1) Crear una nueva cuenta, 2) Realizar un depósito o retiro en una cuenta específica (utilizando polimorfismo), 3) Aplicar el cargo mensual a todas las cuentas (demostrando el uso de la interfaz con verificación de tipo), 4) Salir.

5. Gestión de Animales de un Zoológico

Desarrolla un sistema para clasificar y gestionar animales.

- Interfaz: Comportamiento con métodos comer() y dormir().
- Clase Abstracta: Animal (con atributos especie, nombre, edad) y métodos abstractos emitirSonido() y moverse().
- Clases Concretas: Mamifero (clase intermedia) que hereda de Animal e implementa la interfaz Comportamiento. Mamifero debe implementar los métodos de Comportamiento, pero dejar los de Animal sin definir (es decir, debe ser abstracta si no define todos los métodos abstractos de su padre y de la interfaz, o definir un

cuerpo base para los de la interfaz). Para este ejercicio, haz que Mamifero sea abstracta y defina un método abstracto amamantar().

- Clases Finales: Perro y Gato, ambas heredando de Mamifero. Deben sobreescribir todos los métodos abstractos restantes (incluyendo emitirSonido, moverse, y amamantar).
- Estructura: Usa un ArrayList<Animal> para el inventario.
- Menú: Implementa un menú que permita: 1) Añadir un Perro o Gato (usando un constructor), 2) Hacer que todos los animales emitan su sonido (usando polimorfismo), 3) Mostrar qué animales pueden ser amamantados (demostrando la herencia y el uso del método específico de Mamifero), 4) Salir.