Práctica de Programación Orientada a Objetos – Curso 2024-2025

Nombre: Alejandro

Apellidos: Fernandez Polo

Correo electrónico: afernande7115@alumno.uned.es

Teléfono de contacto: 615172848

Análisis de la Aplicación

La aplicación desarrollada es un sistema de gestión de movilidad que permite a diferentes tipos de usuarios operar una flota de vehículos eléctricos. Se implementan funcionalidades como gestión de usuarios, vehículos, tarifas, mantenimientos y asignaciones. El sistema utiliza los principios de la programación orientada a objetos, herencia por ejemplo, Usuario hereda de Persona, polimorfismo y encapsulamiento.

Las decisiones de diseño incluyen el uso de una clase singleton 'SistemaGestion' para mantener la consistencia global del estado del sistema, clases especializadas como Administrador o Mantenimiento para encapsular comportamientos específicos, y colecciones como ArrayList para almacenar objetos dinámicamente. Se buscó un diseño modular y mantenible.

Apunte: Al introducir las instrucciones a veces no se el motivo pero no se muestran los menos cuando deben, creo que se debe a algo del buffer, pero no he conseguido que siempre funcione perfecto, también soy consciente de que el código no esta comentado en su totalidad por falta de tiempo, pero espero que haya merecido la pena dedicar ese tiempo a controlar todos los errores posibles.

Decisiones de Diseño

- Uso del patrón Singleton en la clase SistemaGestion para garantizar que todas las operaciones del sistema se gestionan desde una única instancia global, evitando inconsistencias en los datos compartidos.
- Separación de responsabilidades: se utilizan clases específicas como GestorAdministrador y GestorMantenimiento para encapsular la lógica de interacción de cada tipo de empleado con el sistema.
- Las entidades principales como Usuario, Vehiculo, Base, Alquiler, Factura están modeladas como clases con estados y comportamientos claramente definidos.
- La elección de ArrayList permite una colección dinámica que se adapta a la creación y eliminación de elementos sin preocupaciones por el tamaño.
- Se emplea la sobrecarga de constructores para permitir diferentes formas de instanciar objetos, como en la clase Alquiler.
- El administrador es el usuario de la aplicación que tiene potestad para crear, editar y eliminar prácticamente cualquier objeto de la aplicación.
- Se ha decidido que las bicicletas y patinetes no tengan coordenadas propias, ya que al siempre tener que estar ligados a bases, estas son las que aportan la localización.
- Por defecto la ciudad tiene unos límites de coordenadas definidos en el constructor de la clase sistemaGestion y que se pueden modificar a gusto.
- Hay un ArrayList para cada tipo de empleado, ya que al principio del desarrollo de la aplicación y sin pensar en su escalabilidad tenía mas sentido, pero soy consciente de que no debería ser así.
- Solo existe un administrador en la aplicación, ya que se ha considerado que este perfil solo realiza cambios en los datos de la aplicación y no se requiere ningún registro de

sus acciones por lo que no es importante que administrador realiza una acción, o lo que es lo mismo no se necesita más de un administrador
- Por defecto cada tipo de vehículo ya tiene una tarifa asignada que se puede modificar, pero no se pueden eliminar ni añadir.

Estrategias Implementadas

- Validación de entradas del usuario a través de Scanner, con manejo de excepciones para evitar errores por entrada inválida.
- Modularidad en la gestión de tareas: cada grupo funcional (usuarios, vehículos, tarifas, etc.) se gestiona desde su propio bloque de menú.
- Asignación de tareas con listas controladas y verificaciones de disponibilidad y duplicados, asegurando la integridad del sistema.
- Evaluación de criterios para promociones de usuarios estándar a premium mediante tres criterios definidos y aplicados con Streams y condiciones.
- Registro y cálculo de importes de alquiler con fechas de inicio y fin usando LocalDateTime y Duration para calcular tiempos exactos de uso.
- Aplicación del encapsulamiento en clases como GestorUsuario, GestorMantenimiento y más
- Uso de enum en algunos datos, con su respectivo control de errores.
- Para una prueba más fácil de la aplicación el constructor de la clase movilidad llama al método `inicializarEjemplos();` pero para la prueba de la aplicación sin datos por defecto solo hay que comentar esta línea de llamada.

Diagrama de Clases

A continuación, se describe detalladamente la relación entre las clases:

1. Clase abstracta Persona:
   - Atributos: dni, nombre, fNacimiento.
   - Métodos: get/set para cada atributo y validadores de fecha y cadena vacía.
   - Subclases:
     a) Usuario.
     b) Empleado.

2. Usuario (abstracta) hereda de Persona:
   - Atributos adicionales: saldo, historial de viajes (ArrayList<Alquiler>).
   - Métodos: get/set para cada atributo y gestionarTareas, agregarSaldo, agregarViaje, alquilarVehiculo, finalizarAlquiler, verHistorialViajes y reportarProblema.
   - Subclases:
     a) UsuarioEstandar: comportamiento básico.
     b) UsuarioPremium: incluye descuento aplicado al importe.

3.GestorUsuario
- Atributos: instancia de SistemaGestion.
- Metodos: gestionarTareasUsuario, mostrarMenuUsuario, agregarSaldo, alquilarVehiculo, finalizarAlquiler y reportarProblema.

4. Empleado (abstracta) hereda de Persona:
   - Subclases:
     a) Administrador: gestión global del sistema.
     b) Mecánico: reparación de bases y vehículos asignados.
     c) Mantenimiento: recogida y reparación de vehículos en mal estado o con batería baja.

5. UsuarioEstandar hereda de Usuario:
- No tiene atributos ni métodos adicionales.

6. UsuarioPremium hereda de Usuario:
- Tiene acceso a descuentos premium.

7. Administrador hereda de Empleado:
- No tiene atributos adicionales.
- Metodos: gestionarTareas, gestionarUsuarios, gestionarVehiculos, gestionarTarifas, gestionarMecanicos, gestionarMantenimientos, gestionarBases, promocionarUsuarioPremium, asignarMantenimiento, asignarMecanico.

8. GestorAdministrador:
- Atributos: instancia de SistemaGestion.
- Métodos: gestionarTareasAdministrador, mostrarMenuAdministrador, gestionarEntidad, gestionarUsuarios, crearUsuario, mostrarUsuarios, eliminarUsuario…

9. Mecánico hereda de Empleado:
- Atributos: instancia de SistemaGestion, vehículos asignados (ArrayList<Vehiculo>), bases asignadas (ArrayList<Base>).
- Métodos: getters, gestionarTareas, agregarVehiculoAsignado, agregaBaseAsignada, verVehiculosAsignados…

10. GestorMecanico:
- Misma estructura que GestorAdministrador y GestorUsuario

11. Mantenimiento hereda de Empleado:
- Atributos: instancia de SistemaGestion, vehículos asignados (ArrayList<Vehiculo>), vehiculos recogidos (ArrayList<Vehiculo>).
- Métodos: getters, gestionarTareas, agregarVehiculoAsignado, agregarVehiculoRecogido, verVehiculosAsignados…

12. GestorMantenimiento:
- Misma estructura que GestorAdministrador y GestorUsuario

13. Vehiculo (abstracta):
  - Atributos: id, nivelBateria, estado.
  - Métodos: calcularConsumoBateria (abstracto), setter y getter.
  - Subclases: Bicicleta, Patinete, Moto.

14. Bicicleta hereda de vehiculo:
- No tiene atributos adicionales
- Implementa el método calcularConsumoBateria

15. Patinete hereda de vehiculo:
- No tiene atributos adicionales
- Implementa el método calcularConsumoBateria

16. Moto hereda de vehiculo:
- Atributos adicionales: coordX, coordY y cilindrada
- Implementa el método calcularConsumoBateria , getters y setters de sus atributos propios.

17. Base:
   - Atributos: id, capacidad, coordX, coordY, vehiculosDisponibles ArrayList<Vehiculo>, huecosDisponibles y tieneFallosMecanicos.
   - Métodos: agregar/eliminar vehículo disponible, getters y setters.

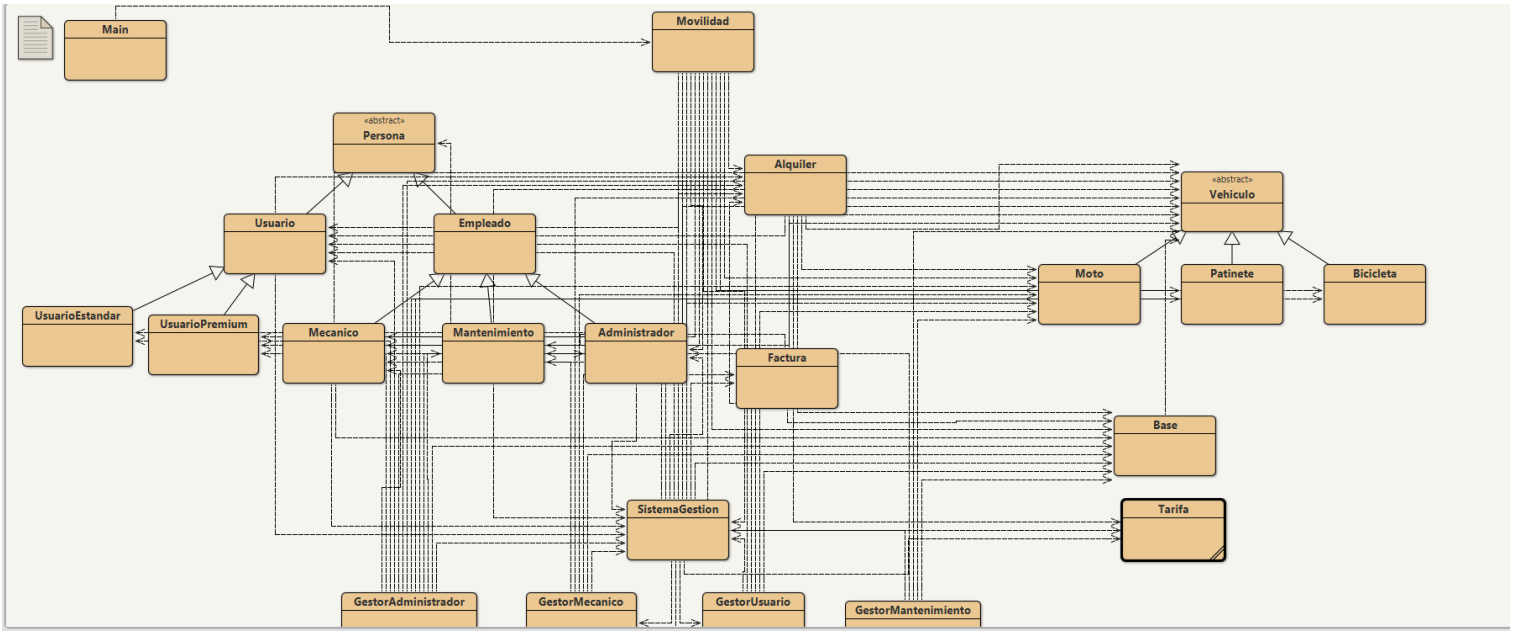
18. Alquiler:
   - Atributos: idAlquiler, vehiculo, usuario, baseInicio, baseFin, coordenadasInicioX, coordenadasInicioY, coordenadasFinX, coordenadasFinY, fechaHoraInicio, fechaHoraFin, tiempoViaje, tarifa, importe.
   - Métodos: getters, setters y finaizarAlquiler.

19. Factura:
   - Atributos: idFactura, mecanico, vehiculo, base, importe, fecha.
   - No tiene métodos adicionales:

20. Tarifa:
   - Atributos: tipoVehiculo, precioPorMinuto, descuentoPremium.
   - Métodos: getters y setters.

21. SistemaGestion:
   - Clase singleton con listas globales: usuarios, vehículos, bases, mantenimientos, etc.
   - Proporciona métodos de operación global como mostrarLista, asignarMantenimiento, gestionarUsuarios, etc.

22. Movilidad:
   - Clase principal que inicia la aplicación



Anexo: Código Fuente

1. Clase Persona:

```
/**
* Clase abstracta que representa una persona.
* Contiene información básica como nombre, DNI y fecha de
nacimiento.
* Proporciona validaciones para asegurar la integridad de los
datos introducidos.
*/
public abstract class Persona {

   // Atributos comunes de cualquier persona
   public String dni;
   public String nombre;
   public int fNacimiento; // Fecha en formato entero AAAAMMDD


   /**
    * Constructor de la clase Persona que inicializa los
atributos de la instancia
    * validando que sean correctos.
    *
    * @param dni          DNI completo, debe tener 8 dígitos y una
letra válida.
    * @param nombre       Nombre de la persona, no puede ser nulo
o vacío.
    * @param fNacimiento Fecha de nacimiento como entero en
formato DDMMAAAA.
    * @throws IllegalArgumentException Si el DNI no es correcto.
    * @throws IllegalArgumentException Si el nombre es nulo o
vacío.
    * @throws IllegalArgumentException Si la fecha de nacimiento
no es válida.
    */
   public Persona(String dni, String nombre, int fNacimiento) {
       if(dniCorrecto(dni)){
           if(contenidoCadena(nombre)){
               if(fechaCorrecta(fNacimiento)){
                   this.dni = dni;
                   this.nombre = nombre;
                   this.fNacimiento = fNacimiento;
               }else{
                   throw new IllegalArgumentException("La fecha
de nacimiento no es correcta");
               }
```

```
            }else{
                throw new IllegalArgumentException("El nombre no
puede ser nulo o vacío");
            }
        }else{
            throw new IllegalArgumentException("El dni
introducido no es correcto");
        }
    }

    // Getters y setters con validaciones
    public String getDni() {
        return dni;
    }

    /**
     * Establece el DNI si es válido.
     *
     * @param dni DNI completo con letra
     * @throws IllegalArgumentException si el DNI no cumple con el
formato y letra
     */
    public void setDni(String dni) {
        if (dniCorrecto(dni)) {
            this.dni = dni;
        } else {
            throw new IllegalArgumentException("El dni
introducido no es correcto");
        }
    }

    public String getNombre() {
        return nombre;
    }

    /**
     * Establece el nombre si no está vacío.
     *
     * @param nombre Nombre de la persona
     * @throws IllegalArgumentException si el nombre es nulo o
está vacío
     */
    public void setNombre(String nombre) {
        if (contenidoCadena(nombre)) {
            this.nombre = nombre;
```

```
        } else {
            throw new IllegalArgumentException("El nombre no
puede ser nulo o vacío");
        }
    }

    public int getfNacimiento() {
        return fNacimiento;
    }

    /**
     * Establece la fecha de nacimiento si es válida.
     *
     * @param fNacimiento Fecha en formato DDMMAAAA
     * @throws IllegalArgumentException si la fecha no es válida
     */
    public void setfNacimiento(int fNacimiento) {
        System.out.println("editando");
        if (fechaCorrecta(fNacimiento)) {
            this.fNacimiento = fNacimiento;
        } else {
            throw new IllegalArgumentException("La fecha de
nacimiento no es correcta");
        }
    }

    /**
     * Valida que una cadena no esté vacía ni contenga solo
espacios.
     *
     * @param cadena Cadena a validar
     * @return true si tiene contenido
     */
    private static boolean contenidoCadena(String cadena) {
        return !cadena.trim().isEmpty();
    }

    /**
     * Valida que el DNI tenga 8 dígitos numéricos seguidos de la
letra correcta.
     *
     * @param dniConLetra DNI completo como string
     * @return true si el formato y la letra son correctos
     */
    private static boolean dniCorrecto(String dniConLetra) {
```

```
        boolean correcto = false;
        char[] arrayDni = dniConLetra.toCharArray();
        if (arrayDni.length == 9) {
            String dni = "";
            int cont = 0;
            for (int j = 0; j < arrayDni.length - 1; j++) {
                if (arrayDni[j] >= '0' && arrayDni[j] <= '9') {
                    dni = dni + arrayDni[j];
                    ++cont;
                }
            }
            if (cont == 8) {
                if (compLetraDni(dniConLetra)) {
                    correcto = true;
                }
            }
        }
        return correcto;
    }

    /**
     * Comprueba si la letra del DNI corresponde a los 8 números
iniciales.
     *
     * @param dni DNI completo como string
     * @return true si la letra es correcta
     */
    private static boolean compLetraDni(String dni) {
        String numDni = dni.substring(0, 8);
        char[] letras = {'T', 'R', 'W', 'A', 'G', 'N', 'Y', 'F',
'P', 'D', 'X',
                'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L',
'C', 'K', 'E'};
        int num = (Integer.parseInt(numDni) % 23);
        String dniCorrecto = numDni + letras[num];
        return dni.equalsIgnoreCase(dniCorrecto);
    }

    /**
     * Valida que la fecha tenga un formato válido y sea una fecha
real.
     *
     * @param fecha Fecha como entero en formato AAAAMMDD
     * @return true si la fecha es válida
     */
```

```java
    private static boolean fechaCorrecta(int fecha) {
        boolean correcta = false;

        int year = fecha % 10000;
        int month = (fecha / 10000) % 100;
        int day = fecha / 1000000;

        if ((month <= 12 && month >= 1) && (year >= 1 && day >=
1)) {
            switch (month) {
                case 1: case 3: case 5: case 6: case 9: case 10:
case 12:
                    correcta = day <= 31;
                    break;
                case 4: case 7: case 8: case 11:
                    correcta = day <= 30;
                    break;
                case 2:
                    correcta = esBisiesto(year) ? day <= 29 : day
<= 28;
                    break;
            }
        }
        return correcta;
    }


    /**
     * Determina si un año es bisiesto.
     *
     * @param year Año a evaluar
     * @return true si es bisiesto
     */
    private static boolean esBisiesto(int year) {
        boolean esBisiesto = false;
        if (year < 400) {
            if (esDivisible(year, 4)) {
                esBisiesto = true;
            }
        } else {
            if ((esDivisible(year, 4) && !esDivisible(year, 100))
                    || (esDivisible(year, 4) && esDivisible(year,
400))) {
                esBisiesto = true;
            }
```

```
        }
        return esBisiesto;
    }

    /**
     * Verifica si un año es divisible por cierto número.
     *
     * @param year Año
     * @param dividendo Divisor
     * @return true si es divisible
     */
    private static boolean esDivisible(int year, int dividendo) {
        return year % dividendo == 0;
    }

    /**
     * Devuelve una representación textual del objeto Persona.
     *
     * @return String con la información principal de la persona
     */
    @Override
    public String toString() {
        return "Persona{" + "nombre=" + nombre + ", dni=" + dni
+ ", fechaNacimiento=" + fNacimiento + '}';
    }

}
```

2. Usuario :

```
import java.util.ArrayList;


/**
* La clase Usuario representa a un usuario en el sistema que
puede realizar diversas operaciones,
* como gestionar tareas, alquilar vehículos y mantener un
historial de alquileres.
* Extiende la clase Persona e incluye atributos y funcionalidades
adicionales,
* específicos de un usuario en el contexto del sistema.
*/
public class Usuario extends Persona {
    /**
```

```
    * Variable que representa una instancia única de la clase
SistemaGestion,
    * utilizada para gestionar las operaciones del sistema
relacionadas con un usuario.
    * Permite realizar acciones como gestionar tareas, alquilar
vehículos
    */
   SistemaGestion sistema = SistemaGestion.getInstancia();
   /**
    * Representa el historial de viajes de un usuario.
    * Esta lista almacena instancias de la clase Alquiler, que
registran los detalles de cada alquiler
    * o viaje realizado por el usuario dentro del sistema.
    * Sirve como registro histórico de los alquileres del
usuario, lo que permite el seguimiento de sus actividades.
    */
   private ArrayList<Alquiler> historialViajes;
   /**
    * Representa el saldo disponible asociado a un usuario en el
sistema.
    * Esta variable se utiliza para rastrear el valor monetario
actual que el usuario puede gastar
    * en actividades como alquilar vehículos o realizar otras
operaciones del sistema.
    */
   public double saldo;


   /**
    * Constructor de la clase Usuario que inicializa una nueva
instancia de usuario con
    * los atributos especificados, incluyendo su saldo inicial y
creando un historial de viajes vacío.
    *
    * @param dni         DNI del usuario, debe ser una cadena de
texto válida con 8 dígitos y una letra.
    * @param nombre      Nombre del usuario, no puede ser nulo ni
vacío.
    * @param fNacimiento Fecha de nacimiento como número entero
en formato DDMMAAAA.
    * @param saldo       Saldo inicial del usuario, expresado
como un valor decimal.
    * @throws IllegalArgumentException Si el DNI, el nombre o la
fecha de nacimiento no son válidos.
    */
```

```java
    public Usuario(String dni, String nombre, int fNacimiento,
double saldo) throws IllegalArgumentException {
        super(dni, nombre, fNacimiento);
        this.saldo = saldo;
        this.historialViajes = new ArrayList<>();
    }

    /**
     * Getters y setters
     */
    public double getSaldo() {
        return saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public ArrayList<Alquiler> getHistorialViajes() {
        return historialViajes;
    }

    public void setHistorialViajes(ArrayList<Alquiler>
historialViajes) {
        this.historialViajes = historialViajes;
    }

    /**
     * Permite a un usuario gestionar las tareas asignadas en el
sistema.
     * Este método delega la gestión de las tareas al sistema,
pasando la instancia del usuario como parámetro.
     */
    public void gestionarTareas() {
        sistema.gestionarTareasUsuario(this);
    }

    /**
     * Permite al usuario agregar saldo a su cuenta a través del
sistema de gestión.
     * Invoca la funcionalidad correspondiente en el sistema para
realizar
     * la operación asociada al usuario actual.
     */
    public void agregarSaldo() {
```

```
        sistema.agregarSaldo(this);
    }

    /**
     * Agrega un nuevo alquiler al historial de viajes del
usuario.
     *
     * @param alquiler el objeto de tipo Alquiler que representa
el nuevo viaje a agregar al historial
     */
    public void agregarViaje(Alquiler alquiler) {
        historialViajes.add(alquiler);
    }

    /**
     * Permite al usuario alquilar un vehículo dentro del sistema
de gestión.
     *
     * Precondiciones:
     * - El usuario debe tener saldo suficiente para realizar el
alquiler.
     *
     * Postcondiciones:
     * - Un nuevo alquiler se registra en el historial del usuario
si la operación
     *   se realiza con éxito.
     * - Los recursos correspondientes (por ejemplo, vehículos) se
bloquean para
     *   el uso de este alquiler.
     */
    public void alquilarVehiculo() {
        sistema.alquilarVehiculo(historialViajes.size(), this);
    }

    /**
     * Finaliza un alquiler en curso utilizando el sistema de
gestión asociado.
     *
     * @param alquiler el alquiler que se desea finalizar.
     */
    public void finalizarAlquiler(Alquiler alquiler) {
        sistema.finalizarAlquiler(alquiler);
    }

    /**
```

```
    * Muestra una lista del historial de viajes del usuario.
    * Este método utiliza la funcionalidad del sistema para
presentar
    * el historial de viajes del usuario de forma clara y con
formato.
    */
   public void verHistorialViajes() {
       sistema.mostrarLista(this.historialViajes, "historial de
viajes");
   }

   /**
    * Informa al sistema sobre un problema detectado por el
usuario.
    * Este método delega la operación al sistema de gestión para
procesar
    * el reporte correspondiente.
    */
   public void reportarProblema() {
       sistema.reportarProblema();
   }

   @Override
   public String toString() {
       return "Persona{" + "nombre=" + nombre + ", dni=" + dni
+ ", fechaNacimiento=" + fNacimiento + ", saldo=" + saldo +
'}';
   }
}
```

3.GestorUsuario

```
import java.util.ArrayList;
import java.util.Scanner;

/**
* La clase GestorUsuario se encarga de gestionar las tareas
relacionadas con el usuario dentro del sistema.
* Proporciona funcionalidad para gestionar las interacciones del
usuario y las operaciones relacionadas,
* como la gestión de alquileres, la adición de fondos y la
notificación de incidencias.
*/
public class GestorUsuario {

   Scanner scanner = new Scanner(System.in);
```

```
    SistemaGestion sistema;

    public GestorUsuario(SistemaGestion sistema) {
        this.sistema = sistema;
    }

    /**
     * Gestiona las tareas del usuario en el sistema, permitiendo
realizar diversas acciones
     * como agregar saldo, alquilar vehículos, reportar problemas,
ver el historial de viajes
     * o finalizar un alquiler activo.
     *
     * @param usuario Instancia del usuario que realizará las
acciones en el sistema.
     *                El usuario debe estar registrado previamente
en el sistema y su información
     *                debe ser válida para realizar operaciones.
     */
    public void gestionarTareasUsuario(Usuario usuario) {
        String accion;
        do {

System.out.println("\n----------------------------------------
------------------------------------------------\n");
            System.out.println("Bienvenido " +
usuario.getNombre() + ", ¿qué desea hacer?");
            // Buscar alquiler no finalizado
            Alquiler alquilerNoFinalizado =
usuario.getHistorialViajes().stream()
                    .filter(a -> a.getImporte() == 0)
                    .findFirst()
                    .orElse(null);
            mostrarMenuUsuario(usuario.getNombre(),
alquilerNoFinalizado != null);
            accion = scanner.nextLine().trim();
            if (accion.isEmpty()) {
                System.out.println("Saliendo...");
            } else {
                switch (accion) {
                    case "1":
                        usuario.agregarSaldo();
                        break;
                    case "2":
                        usuario.alquilarVehiculo();
```

```
                        break;
                    case "3":
                        usuario.reportarProblema();
                        break;
                    case "4":
                        usuario.verHistorialViajes();
                        break;
                    case "5":
                        if (alquilerNoFinalizado != null) {

usuario.finalizarAlquiler(alquilerNoFinalizado);
                        } else {
                            System.out.println("No hay alquiler
activo para finalizar.");
                        }
                        break;
                    default:
                        System.out.println("No ha introducido una
opción correcta");
                }
            }

        } while (!accion.trim().isEmpty());
    }

    /**
     * Muestra un menú con diferentes opciones para el usuario
según la información de su cuenta
     * y el estado del alquiler.
     *
     * @param nombreUsuario: el nombre del usuario para el que se
muestra el menú.
     * @param hayAlquilerActivo: un valor booleano que indica si
el usuario tiene un alquiler activo.
     */
    private void mostrarMenuUsuario(String nombreUsuario, boolean
hayAlquilerActivo) {
        System.out.println("1- Añadir saldo");
        System.out.println("2- Alquilar un vehículo");
        System.out.println("3- Reportar un problema");
        System.out.println("4- Ver historial de viajes");
        if (hayAlquilerActivo) {
            System.out.println("5- Finalizar alquiler activo");
        }
        System.out.println("Pulse Enter para salir\n");
```

```
    }

    /**
     * Agrega saldo a la cuenta de un usuario especificado.
     * Solicita al usuario un monto a agregar, valida que sea
positivo
     * y lo suma al saldo actual del usuario. Si el monto
ingresado no
     * es válido o es negativo, se informa al usuario con un
mensaje de error.
     *
     * @param usuario El objeto de tipo Usuario al que se le desea
agregar saldo.
     *                Debe ser una instancia válida y no nula.
     */
    public void agregarSaldo(Usuario usuario) {
        System.out.println("Introduzca el saldo que desea añadir:
");
        try {
            double saldo =
Double.parseDouble(scanner.nextLine());
            if (saldo < 0) {
                System.out.println("El saldo a agregar debe ser
un número entero positivo");
            } else {
                usuario.setSaldo(usuario.getSaldo() + saldo);
                System.out.println("Saldo agregado
correctamente.");
                System.out.println(usuario.getNombre() + " tu
saldo actual es de " + usuario.getSaldo() + "€");
            }
        } catch (IllegalArgumentException e) {
            System.out.println("Error al añadir el saldo: " +
e.getMessage());
        }
    }

    /**
     * Permite a un usuario alquilar un vehículo del sistema, el
cual puede ser una bicicleta,
     * un patinete o una motocicleta. El método solicita al
usuario seleccionar el tipo
     * de vehículo, la ubicación del vehículo a alquilar, y las
coordenadas de destino.
     *
```

```
    * @param idAlquiler Identificador único del alquiler que se
va a generar.
    * @param usuario Usuario que desea realizar el alquiler del
vehículo.
    * @return Un objeto de tipo Alquiler que representa el
alquiler realizado, o null si el
    *         alquiler no pudo efectuarse por falta de saldo,
vehículos disponibles o por
    *         coordenadas fuera de los límites.
    */
   public Alquiler alquilarVehiculo(int idAlquiler, Usuario
usuario) {
       if (sistema.getVehiculos().isEmpty()) {
           System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
           System.out.println("No hay vehiculos en el sistema");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
       } else {
           Alquiler alquiler = null;
           if (usuario.getSaldo() < 0) {
               System.out.println("No tienes saldo suficiente
para alquilar el vehiculo.");
           } else {
               String accion = "1";
               while (!accion.trim().isEmpty() && alquiler ==
null) {
                   System.out.println('\n' +
"-------------------------------------------------------------
---------------------------" + '\n');
                   System.out.println("¿Qué vehiculo desea
alquilar?" + '\n' +
                           "1- Bicicleta " + '\n' +
                           "2- Patinete " + '\n' +
                           "3- Motocicleta " + '\n' +
                           "Pulse Enter para salir" + '\n');
                   accion = scanner.nextLine().trim();
                   switch (accion) {
                       case "1" -> {
                           ArrayList<Base> basesdisp = new
ArrayList();
                           for (Base b : sistema.getBases()) {
```

```
                                if
(!b.getBicicletasDisponibles().isEmpty()) {
                                    basesdisp.add(b);
                                }
                            }
                            if (basesdisp.isEmpty()) {
                                System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
                                System.out.println("No hay
bicicletas en el sistema");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
                            }
                        }
                        case "2" -> {
                            ArrayList<Base> basesdisp = new
ArrayList();
                            for (Base b :
sistema.getBasesDisponibles()) {
                                if
(!b.getPatinetesDisponibles().isEmpty()) {
                                    basesdisp.add(b);
                                }
                            }
                            if (basesdisp.isEmpty()) {
                                System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
                                System.out.println("No hay
patinetes en el sistema");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
                            }
                        }
                        case "3" -> {
                            ArrayList<Vehiculo> motosDisp = new
ArrayList();
                            for (Vehiculo v :
sistema.getVehiculos()) {
                                if (v instanceof Moto) {
                                    motosDisp.add(v);
                                }
```

```
                            }
                            if (motosDisp.isEmpty()) {
                                System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
                                System.out.println("No hay motos
en el sistema");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
                            }
                        }
                    }
                    int coordX = -1;
                    int coordY = -1;
                    int coordXFin = -1;
                    int coordYFin = -1;
                    while (coordX == -1 || coordY == -1 ||
coordXFin == -1 || coordYFin == -1) {
                        System.out.println("Para encontar un
vehiculo cerca de usted introduzca sus coordenadas, primero su
coordenada X");
                        try {
                            coordX =
Integer.parseInt(scanner.nextLine());
                            System.out.println("Ahora su
coordenada Y");
                            coordY =
Integer.parseInt(scanner.nextLine());
                            System.out.println("Introduzca a
donde se dirige, primero la coordenada X");
                            coordXFin =
Integer.parseInt(scanner.nextLine());
                            System.out.println("Ahora la
coordenada Y");
                            coordYFin =
Integer.parseInt(scanner.nextLine());
                            if (coordXFin < 0 || coordXFin >
sistema.getLimiteX() || coordYFin < 0 || coordYFin >
sistema.getLimiteY()) {
                                System.out.println("Las
coordenadas a las que se dirije están fuera de los límites de
la ciudad");
                                return null;
                            }
```

```
                    } catch (NumberFormatException e) {
                        System.err.println("Entrada
inválida.");
                    }
                }
                try {
                    switch (accion) {
                        case "1" -> {
                            ArrayList<Base> basesdisp = new
ArrayList();
                            for (Base b :
sistema.getBases()) {
                                if
(!b.getBicicletasDisponibles().isEmpty()) {
                                    basesdisp.add(b);
                                }
                            }
                            basesdisp =
sistema.ordenarBases(basesdisp, coordX, coordY);
                            System.out.println("Este es el
listado de las bases mas cercanas con bicicletas disponibles:
");
                            for (Base b : basesdisp) {
                                System.out.println(b);
                            }

                            Base baseEscogida = null;
                            while (baseEscogida == null) {
                                System.out.println('\n' +
"Seleccione el id de la base de la que va a alquilar su bici: "
+ '\n');
                                int id =
Integer.parseInt(scanner.nextLine());
                                for (Base b : basesdisp) {
                                    if (b.getId() == id) {
                                        baseEscogida = b;
                                    }
                                }
                                if (baseEscogida != null) {
                                    Vehiculo biciEscogida =
null;
                                    while (biciEscogida ==
null) {
```

```
System.out.println("La base elegida tiene estas bicicletas
disponibles, introduzca el id de " +
                                                    "la
bicicleta que quiere alquilar: ");
                                            for (Vehiculo bici :
baseEscogida.getBicicletasDisponibles()) {

System.out.println(bici);
                                            }
                                            int idBici =
Integer.parseInt(scanner.nextLine());
                                            for (Vehiculo v :
baseEscogida.getBicicletasDisponibles()) {
                                                if (v.getId() ==
idBici) {
                                                    biciEscogida
= v;
                                                }
                                            }
                                            if (biciEscogida !=
null) {
                                                alquiler = new
Alquiler(idAlquiler, biciEscogida, usuario, baseEscogida,
sistema.ordenarBases(sistema.getBasesDisponiblesConHueco(),
coordXFin, coordYFin).get(0), 0, 0, 0, 0,
sistema.tarifaBicicleta);

baseEscogida.eliminarVehiculoDisponible(biciEscogida);

biciEscogida.setEstado("RESERVADO");

usuario.agregarViaje(alquiler);

System.out.println("Alquiliando bicicleta " + biciEscogida);
                                                break;
                                            } else {

System.out.println("No se ha encontrado la bicicleta con id " +
idBici);
                                            }
                                        }

                                    } else {
```

```
                                        System.out.println("No se
ha encontrado la base con id " + id);
                                    }
                                }

                            }
                            case "2" -> {
                                ArrayList<Base> basesdisp = new
ArrayList();
                                for (Base b :
sistema.getBasesDisponibles()) {
                                    if
(!b.getPatinetesDisponibles().isEmpty()) {
                                        basesdisp.add(b);
                                    }
                                }
                                basesdisp =
sistema.ordenarBases(basesdisp, coordX, coordY);
                                System.out.println("Este es el
listado de las bases mas cercanas con patinetes disponibles:
");
                                for (Base b : basesdisp) {
                                    System.out.println(b);
                                }

                                Base baseEscogida = null;
                                while (baseEscogida == null) {
                                    System.out.println('\n' +
"Seleccione el id de la base de la que va a alquilar su
patinete: " + '\n');
                                    int id =
Integer.parseInt(scanner.nextLine());
                                    for (Base b : basesdisp) {
                                        if (b.getId() == id) {
                                            baseEscogida = b;
                                        }
                                    }
                                    if (baseEscogida != null) {
                                        Vehiculo patinEscogido =
null;

                                        while (patinEscogido ==
null) {

System.out.println("La base elegida tiene estos patinetes
disponibles, introduzca el id de " +
```

```
                                                    "el patinete
que quiere alquilar: ");
                                                for (Vehiculo patin
: baseEscogida.getPatinetesDisponibles()) {

System.out.println(patin);
                                                }
                                                int idPatin =
Integer.parseInt(scanner.nextLine());
                                                for (Vehiculo v :
baseEscogida.getPatinetesDisponibles()) {
                                                    if (v.getId() ==
idPatin) {

patinEscogido = v;
                                                    }
                                                }
                                                if (patinEscogido !=
null) {
                                                    alquiler = new
Alquiler(idAlquiler, patinEscogido, usuario, baseEscogida,
sistema.ordenarBases(sistema.getBasesDisponiblesConHueco(),
coordXFin, coordYFin).get(0), 0, 0, 0, 0,
sistema.tarifaPatinete);

baseEscogida.eliminarVehiculoDisponible(patinEscogido);

patinEscogido.setEstado("RESERVADO");

usuario.agregarViaje(alquiler);

System.out.println("Alquiliando patinete " + patinEscogido);
                                                    break;
                                                } else {

System.out.println("No se ha encontrado el patinete con id " +
idPatin);
                                                }
                                            }

                                        } else {
                                            System.out.println("No se
ha encontrado la base con id " + id);
                                        }
                                    }
```

```
                            }
                            case "3" -> {
                                ArrayList<Vehiculo> motosDisp =
new ArrayList();
                                for (Vehiculo v :
sistema.getVehiculos()) {
                                    if (v instanceof Moto &&
v.getEstado().equals(EstadosVehiculo.DISPONIBLE) &&
v.getNivelBateria() > 20) {
                                        motosDisp.add(v);
                                    }
                                }
                                if (motosDisp.isEmpty()) {
                                    System.out.println('\n' +
"-----------------------------------------------------------
---------------------------");
                                    System.out.println("No hay
motos disponibles en el sistema");

System.out.println("-----------------------------------------
---------------------------------------------" + '\n');
                                    return null;
                                }else {
                                    motosDisp =
sistema.ordenarMotos(motosDisp, coordX, coordY);
                                    Vehiculo motoEscogida =
null;
                                    while (motoEscogida == null)
{
                                        System.out.println("Este
es el listado de las motos mas cercanas, introduzca el id de la
que desea alquilar: ");
                                        for (Vehiculo m :
motosDisp) {

System.out.println(m);
                                        }
                                        int idMoto =
Integer.parseInt(scanner.nextLine());
                                        for (Vehiculo v :
motosDisp) {
                                            if (v.getId() ==
idMoto) {
                                                motoEscogida =
v;
```

```
                                                }
                                            }
                                            if (motoEscogida !=
null) {
                                                alquiler = new
Alquiler(idAlquiler, motoEscogida, usuario, null, null, coordX,
coordY, coordXFin, coordYFin, sistema.tarifaPatinete);

motoEscogida.setEstado("RESERVADO");

usuario.agregarViaje(alquiler);

System.out.println("Alquiliando moto " + motoEscogida);
                                                break;
                                            } else {

System.out.println("No se ha encontrado la moto con id " +
idMoto);
                                            }
                                        }
                                    }
                                }
                                default ->
System.out.println("Saliendo...");
                            }
                        } catch (NumberFormatException e) {
                            System.err.println("Entrada inválida.");
                        }
                    }
                }
                return alquiler;
            }
            return null;
        }

        /**
         * Finaliza un proceso de alquiler en curso.
         *
         * @param alquiler El objeto de alquiler que contiene los
detalles necesarios del proceso de alquiler,
         * incluyendo el vehículo asociado, el usuario y la
información del alquiler.
         */
        public void finalizarAlquiler(Alquiler alquiler) {
            alquiler.finalizarAlquiler();
```

```
        Vehiculo vehiculo = alquiler.getVehiculo();
        vehiculo.setEstado("disponible");

vehiculo.calcularConsumoBateria(alquiler.getTiempoViaje());
        Usuario usuario = alquiler.getUsuario();
        usuario.setSaldo(usuario.getSaldo() -
alquiler.getImporte());
        if (vehiculo instanceof Moto) {
            ((Moto)
vehiculo).setCoordX(alquiler.getCoordenadasFinX());
            ((Moto)
vehiculo).setCoordY(alquiler.getCoordenadasFinY());
        } else {

alquiler.getBaseFin().agregarVehiculoDisponible(vehiculo);
        }
        System.out.println("Finalizando alquiler...");
        System.out.println("Tu saldo restante es de " +
alquiler.getUsuario().getSaldo() + "€");

    }

    /**
     * Este método permite al usuario reportar problemas
relacionados con vehículos o bases dentro del sistema.
     * Los usuarios pueden optar por reportar un vehículo o una
base con mal funcionamiento.
     *
     * El método también gestiona correctamente las entradas
numéricas no válidas y ofrece una opción para salir.
     *
     * Excepciones:
     * - Se gestiona la excepción NumberFormatException si se
proporciona una entrada no numérica para los ID de vehículos o
bases.
     */
    public void reportarProblema() {
        try {
            String accion = "1";
            while (!accion.trim().isEmpty()) {
                System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
                System.out.println("Quiere reportar un problema,
de que se trata:" + '\n' +
```

```
                        "1- Vehiculo estropeado " + '\n' +
                        "2- Base estropeada " + '\n' +
                        "Pulse Enter para salir" + '\n');

                accion = scanner.nextLine().trim();
                if (!accion.trim().isEmpty()) {
                    switch (accion) {
                        case "1":
                            if
(sistema.getVehiculos().isEmpty()) {
                                System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
                                System.out.println("No hay
vehiculos en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
                            }else{

sistema.mostrarLista(sistema.getVehiculos(), "vehiculos");
                                Vehiculo vehiculoReportar =
null;
                                while (vehiculoReportar == null)
{

System.out.println("Introduzca el id del vehiculo que desea
reportar: ");
                                    int idVehiculo =
Integer.parseInt(scanner.nextLine());
                                    for (Vehiculo v :
sistema.getVehiculos()) {
                                        if (v.getId() ==
idVehiculo) {
                                            vehiculoReportar =
v;
                                        }
                                    }
                                    if (vehiculoReportar ==
null) {
                                        System.out.println("No se
ha encontrado el vehiculo con id " + idVehiculo);
                                    } else {

vehiculoReportar.setEstado("averiado");
```

```
                                        System.out.println("El
vehiculo con id " + vehiculoReportar.getId() + " ha sido
reportado");
                                    }
                                }
                            }
                            break;
                        case "2":
                            if (sistema.getBases().isEmpty()) {
                                System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
                                System.out.println("No hay bases
en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
                            }else{

sistema.mostrarLista(sistema.getBases(), "bases");
                                Base baseReportar = null;
                                while (baseReportar == null) {

System.out.println("Introduzca el id de la base que desea
reportar: ");
                                    int idBase =
Integer.parseInt(scanner.nextLine());
                                    for (Base b :
sistema.getBases()) {
                                        if (b.getId() == idBase)
{
                                            baseReportar = b;
                                        }
                                    }
                                    if (baseReportar == null) {
                                        System.out.println("No se
ha encontrado la base con id " + idBase);
                                    } else {

baseReportar.setTieneFallosMecanicos(true);
                                        System.out.println("La
base con id " + baseReportar.getId() + " ha sido reportada");
                                    }
                                }
                            }
```

```
                            break;
                        default:
                            System.out.println("No ha introducido
una opción correcta");
                    }
                } else {
                    System.out.println("Saliendo...");
                }
            }
        } catch (NumberFormatException e) {
            System.err.println("Entrada inválida.");
        }
    }
}
```

4. Empleado:

```
/**
* Representa a un empleado dentro del sistema, heredando de la
clase Persona.
* Esta clase sirve como clase base para roles específicos de
empleado,
* como administradores, mecánicos y personal de mantenimiento.
*/
public class Empleado extends Persona
{

    /**
     * Construye una instancia de la clase Empleado inicializando
sus atributos
     * usando los parámetros proporcionados. Este constructor
llama al constructor de la superclase
     * para validar e inicializar las propiedades compartidas de
una Persona.
     *
     * @param dni El DNI completo del empleado. Debe constar de 8
dígitos y una letra válida.
     * @param nombre El nombre del empleado. No puede ser nulo ni
estar vacío.
     * @param fNacimiento La fecha de nacimiento del empleado como
un entero con el formato DDMMAAAA.
     * @throws IllegalArgumentException Si el DNI no es válido o
tiene un formato incorrecto.
     * @throws IllegalArgumentException Si el nombre es nulo o
está vacío.
     * @throws IllegalArgumentException Si la fecha de nacimiento
no es válida.
```

```
    */
   public Empleado(String dni, String nombre, int fNacimiento)
throws IllegalArgumentException
   {
       super(dni, nombre, fNacimiento);
   }

}
```

5. UsuarioEstandar:

```
/**
* La clase UsuarioEstandar es una subclase de la clase Usuario
que representa a un usuario estándar
* dentro del sistema. Hereda los atributos y comportamiento de la
clase Usuario y no añade nuevas funcionalidades,
* pero permite crear instancias específicas para usuarios que
pertenecen a esta categoría.
*
* Dado que extiende a Usuario, tiene acceso a todas las
propiedades y métodos inherentes, como el manejo
* de saldos, historial de viajes y las interacciones con el
sistema de gestión.
*/
public class UsuarioEstandar extends Usuario
{

   /**
    * Constructor de la clase UsuarioEstandar que inicializa una
nueva instancia de usuario estándar
    * con los atributos especificados. Este constructor invoca al
constructor de la superclase Usuario.
    *
    * @param dni         DNI del usuario, debe ser una cadena de
texto válida con 8 dígitos y una letra.
    * @param nombre      Nombre del usuario, no puede ser nulo ni
vacío.
    * @param fNacimiento Fecha de nacimiento del usuario como
número entero en formato DDMMAAAA.
    * @param saldo       Saldo inicial del usuario estándar,
expresado como un valor decimal.
```

```
    */
   public UsuarioEstandar(String dni, String nombre, int
fNacimiento, double saldo)
   {
       super(dni, nombre, fNacimiento, saldo);
   }

   @Override
   public String toString() {
       return "UsuarioEstandar{" +
               "dni='" + dni + '\'' +
               ", nombre='" + nombre + '\'' +
               ", fNacimiento=" + fNacimiento +
               ", saldo=" + saldo +
               '}';
   }
}
```

6. UsuarioPremium:

```
/**
* La clase UsuarioPremium extiende las funcionalidades de la
clase Usuario
* para representar a un usuario con privilegios avanzados dentro
del sistema.
* Este tipo de usuario tiene acceso a características exclusivas
o ventajas añadidas
* en comparación con los usuarios estándar.
*/
public class UsuarioPremium extends Usuario
{

   /**
    * Constructor de la clase UsuarioPremium que crea una nueva
instancia de usuario premium
    * con los atributos especificados.
    *
    * @param dni         DNI del usuario, debe ser una cadena de
texto válida con 8 dígitos y una letra.
    * @param nombre      Nombre del usuario, no puede ser nulo ni
vacío.
    * @param fNacimiento Fecha de nacimiento del usuario como
número entero en formato DDMMAAAA.
    * @param saldo       Saldo inicial del usuario premium,
expresado como un valor decimal.
```

```
     */
    public UsuarioPremium(String dni, String nombre, int
fNacimiento, double saldo)
    {
        super(dni, nombre, fNacimiento, saldo);
    }

    @Override
    public String toString() {
        return "UsuarioPremium{" +
                "dni='" + dni + '\'' +
                ", nombre='" + nombre + '\'' +
                ", fNacimiento=" + fNacimiento +
                ", saldo=" + saldo +
                '}';
    }
}
```

7. Administrador:

```
/**
* La clase Administrador representa un empleado con
responsabilidades específicas
* para gestionar diversas áreas dentro del sistema. Esta clase
extiende
* la funcionalidad básica de un empleado, proporcionándole
capacidades avanzadas
* de gestión mediante métodos especializados.
*/
public class Administrador extends Empleado {

    SistemaGestion sistema = SistemaGestion.getInstancia();

    /**
     * Constructor de la clase Administrador que inicializa una
nueva instancia con los
     * valores proporcionados. Este constructor llama al
constructor de la superclase
     * Empleado para validar e inicializar los atributos básicos.
     *
     * @param dni El DNI completo del administrador. Debe constar
de 8 dígitos y una letra válida.
     * @param nombre El nombre del administrador. No puede ser
nulo ni estar vacío.
```

```
    * @param fNacimiento La fecha de nacimiento del administrador
como un entero con el formato DDMMAAAA.
    * @throws IllegalArgumentException Si el DNI no es válido o
tiene un formato incorrecto.
    * @throws IllegalArgumentException Si el nombre es nulo o
está vacío.
    * @throws IllegalArgumentException Si la fecha de nacimiento
no es válida.
    */
   public Administrador(String dni, String nombre, int
fNacimiento) throws IllegalArgumentException {
       super(dni, nombre, fNacimiento);
   }

   /**
    * Este método permite a un administrador gestionar tareas
relacionadas
    * con su ámbito de responsabilidad dentro del sistema.
    * Al invocar este método, se redirige la implementación al
sistema de gestión,
    * el cual ejecuta las acciones correspondientes a nivel
administrativo.
    *
    * El método funciona como un intermediario, invocando la
funcionalidad
    * del sistema de gestión con el contexto del administrador
actual.
    */
   public void gestionarTareas() {
       sistema.gestionarTareasAdministrador(this);
   }

   /**
    * Permite al administrador gestionar usuarios en el sistema.
    * Este método delega la funcionalidad de gestión de usuarios
a la capa de gestión del sistema.
    */
   public void gestionarUsuarios() {
       sistema.gestionarUsuarios();
   }

   /**
    * Gestiona los vehículos a través del sistema de gestión.
Este método delega la
```

```
    * operación al subsistema correspondiente encargado de las
funcionalidades
    * relacionadas con la gestión y administración de los
vehículos.
    */
   public void gestionarVehiculos() {
       sistema.gestionarVehiculos();
   }

   /**
    * Permite al administrador gestionar las tarifas disponibles
dentro del sistema.
    * Este método delega la gestión de tarifas al sistema
asociado, permitiendo que
    * se realicen las operaciones necesarias relacionadas con las
tarifas establecidas.
    */
   public void gestionarTarifas() {
       sistema.gestionarTarifas();
   }

   /**
    * Este método se encarga de gestionar las mecánicas dentro
del sistema.
    * Delega la funcionalidad de gestión al componente de gestión
integradadel sistema,
    * lo que permite al administrador supervisar o configurar las
operaciones relacionadas con las mecánicas.
    */
   public void gestionarMecanicos() {
       sistema.gestionarMecanicos();
   }

   /**
    * Este método permite al administrador gestionar los procesos
de mantenimiento dentro del sistema.
    * Delega la lógica relacionada con el mantenimiento a la capa
de gestión del sistema.
    * Este método suele garantizar que las operaciones de
mantenimiento se gestionen y
    * supervisen eficientemente, incluyendo acciones como la
programación, la asignación
    * de recursos.
    */
   public void gestionarMantenimientos() {
```

```
        sistema.gestionarMantenimientos();
    }

    /**
     * Invoca la funcionalidad de gestión de bases del sistema.
Este método actúa como
     * un punto de delegación para gestionar las operaciones
relacionadas con la gestión
     * de bases dentro del sistema. Los detalles de las
operaciones se implementan
     * dentro de la lógica de gestión del sistema.
     */
    public void gestionarBases() {
        sistema.gestionarBases();
    }

    /**
     * Promueve a un usuario a la categoría de usuario premium
dentro del sistema.
     * Este método delega la acción a la instancia del sistema de
gestión asociada
     * al administrador, encargándose de realizar las operaciones
necesarias para
     * actualizar el estado del usuario según las reglas definidas
en el sistema.
     */
    public void promocionarUsuarioPremium(){
        sistema.promocionarUsuarioPremium();
    }

    /**
     * Asigna tareas de mantenimiento a las entidades
correspondientes dentro del sistema de gestión.
     * Este método es utilizado por la clase Administrador para
delegar responsabilidades de mantenimiento.
     */
    public void asignarMantenimiento(){
        sistema.asignarMantenimiento();
    }

    /**
     * Asigna un mecánico para la resolución de problemas o tareas
de mantenimiento detectadas
     * en el sistema. Este método delega la acción a una instancia
del sistema de gestión, que
```

```
    * centraliza las operaciones de asignación de mecánicos a las
áreas o vehículos que lo requieran.
    */
   public void asignarMecanico(){
       sistema.asignarMecanico();
   }

}
```

8. GestorAdministrador:

```
import java.time.LocalDateTime;
import java.util.*;

/**
* La clase GestorAdministrador se encarga de gestionar las
operaciones relacionadas con la administración
* de un sistema de gestión. Proporciona funcionalidades para
manejar entidades, usuarios, vehículos, tarifas,
* mecánicos, empleado de mantenimiento, bases y otros aspectos
organizativos y operativos.
*/
public class GestorAdministrador {

   Scanner scanner = new Scanner(System.in);
   SistemaGestion sistema;


   public GestorAdministrador(SistemaGestion sistema) {
       this.sistema = sistema;
   }

   /**
    * Gestiona las tareas administrativas del sistema presentando
un menú de opciones
    * disponibles para el administrador y ejecutando las acciones
correspondientes según la
    * elección del usuario.
    *
```

```
    * @param administrador El objeto administrador que ejecutará
las tareas. Este
    * parámetro se utiliza para invocar diversos métodos de
gestión, como la gestión
    * de usuarios, vehículos, tarifas, tareas de mantenimiento,
etc.
    */
   public void gestionarTareasAdministrador(Administrador
administrador) {

System.out.println("\n-----------------------------------------
-----------------------------------------\n");
       System.out.println("Bienvenido " +
administrador.getNombre() + " al sistema de gestión de
movilidad sostenible.");
       String accion;

       do {
           mostrarMenuAdministrador();
           accion = scanner.nextLine().trim();

           if (accion.isEmpty()) {
               System.out.println("Saliendo...");
           } else {
               switch (accion) {
                   case "1":
                       administrador.gestionarUsuarios();
                       break;
                   case "2":
                       administrador.gestionarVehiculos();
                       break;
                   case "3":
                       administrador.gestionarTarifas();
                       break;
                   case "4":
                       administrador.gestionarMecanicos();
                       break;
                   case "5":
                       administrador.gestionarMantenimientos();
                       break;
                   case "6":
                       administrador.gestionarBases();
                       break;
                   case "7":
```

```
administrador.promocionarUsuarioPremium();
                        break;
                    case "8":
                        administrador.asignarMantenimiento();
                        break;
                    case "9":
                        administrador.asignarMecanico();
                        break;
                    default:
                        System.out.println("No ha introducido una
opción correcta");
                }
            }
        } while (!accion.trim().isEmpty());
    }

    /**
     * Muestra un menú con opciones administrativas disponibles
para el administrador.
     * Este menú permite al administrador gestionar diversas
entidades y asignaciones,
     * proporcionando funcionalidades como la gestión de usuarios,
vehículos y tarifas,
     * y más. Las opciones se enumeran numéricamente e incluyen la
función de salida.
     */
    private void mostrarMenuAdministrador() {
        System.out.println("¿Qué desea realizar?");
        System.out.println("1- Gestionar usuarios");
        System.out.println("2- Gestionar vehículos");
        System.out.println("3- Gestionar tarifas");
        System.out.println("4- Gestionar mecánicos");
        System.out.println("5- Gestionar mantenimientos");
        System.out.println("6- Gestionar bases");
        System.out.println("7- Promocionar usuarios");
        System.out.println("8- Asignar vehículo a empleado de
mantenimiento");
        System.out.println("9- Asignar vehículo o base a
mecánico");
        System.out.println("Pulse Enter para salir\n");
    }

    /**
```

```
    * Método genérico para gestionar cualquier tipo de entidad
proporcionando
    * diferentes opciones como crear, mostrar, editar o eliminar.
    *
    * @param tipoEntidad Descripción textual del tipo de entidad
que se está gestionando (por ejemplo, "Usuario", "Vehículo").
    * @param crear Acción asociada a la creación de una nueva
instancia de la entidad.
    * @param mostrar Acción asociada a la visualización de todas
las entidades existentes.
    * @param editar Acción asociada a la modificación de una
entidad específica.
    * @param eliminar Acción asociada a la eliminación de una
entidad específica.
    * @param opciones Array de cadenas que contiene opciones
adicionales que pueden ser utilizadas para personalizar
    *                 el comportamiento del menú o del proceso de
gestión, aunque en este método no están directamente utilizadas.
    */
   // Método genérico para gestionar cualquier tipo de entidad
   public void gestionarEntidad(String tipoEntidad,
                                Runnable crear,
                                Runnable mostrar,
                                Runnable editar,
                                Runnable eliminar,
                                String[] opciones) {
       String accion = "5";
       while (!accion.trim().isEmpty()) {
           System.out.println('\n' +
"------------------------------------------------------------
---------------------------" + '\n');
           System.out.println("¿Qué desea hacer con los " +
tipoEntidad + "s?" + '\n' +
                   "1- Añadir un " + tipoEntidad + '\n' +
                   "2- Mostrar los " + tipoEntidad + "s" + '\n'
+
                   "3- Editar un " + tipoEntidad + '\n' +
                   "4- Eliminar un " + tipoEntidad + '\n' +
                   "Pulse Enter para salir" + '\n');
           accion = scanner.nextLine().trim();
           if (!accion.trim().isEmpty()) {
               switch (accion) {
                   case "1":
                       crear.run();
                       break;
```

```
                    case "2":
                        mostrar.run();
                        break;
                    case "3":
                        editar.run();
                        break;
                    case "4":
                        eliminar.run();
                        break;
                    default:
                        System.out.println("No ha introducido una
opción correcta");
                }
            } else {
                System.out.println("Saliendo...");
            }
        }
    }

    /**
     * Gestiona las operaciones relacionadas con los usuarios,
incluyendo la creación, visualización,
     * edición y eliminación de usuarios.
     * Este método utiliza un mecanismo genérico de gestión de
entidades mediante el método gestionarEntidad.
     *
     * Los métodos internos como crearUsuario, mostrarUsuarios,
editarUsuario y eliminarUsuario
     * se utilizan para realizar tareas específicas.
     */
    public void gestionarUsuarios() {
        gestionarEntidad("usuario",
                this::crearUsuario,
                this::mostrarUsuarios,
                this::editarUsuario,
                this::eliminarUsuario,
                new String[]{"1", "2", "3", "4"});
    }

    /**
     * Método que permite crear un nuevo usuario estándar dentro
del sistema.
     * Este método solicita al administrador que introduzca los
datos necesarios,
```

```
    * como el DNI, nombre, fecha de nacimiento y saldo inicial
del usuario.
    * Si ya existe un usuario con el mismo DNI, el nuevo usuario
no será agregado.
    *
    * En su implementación, este método utiliza una estructura
genérica que
    * permite la creación de una persona, delegando parte de la
construcción
    * de la instancia del usuario estándar a un*/
   public void crearUsuario() {
       crearPersona("usuario", (dni, nombre, fechaNac) -> {
           System.out.println("Introduce el saldo del usuario");
           double saldo = -1;
           try {
               saldo = Double.parseDouble(scanner.nextLine());
           } catch (IllegalArgumentException e) {
               System.out.println("Error al añadir el saldo: " +
e.getMessage());
           }
           UsuarioEstandar usuario = new UsuarioEstandar(dni,
nombre, fechaNac, saldo);
           boolean agregado = false;
           for (Usuario u : sistema.getUsuarios()) {
               if
(u.getDni().equalsIgnoreCase(usuario.getDni())) {
                   agregado = true;
               }
           }
           if (!agregado) {
               sistema.getUsuarios().add(usuario);
               System.out.println("Usuario agregado");
           } else {
               System.out.println("Ya existe un usuario con ese
dni, el usuario no se ha podido agregar");
           }

       });
   }

   /**
    * Muestra la lista de usuarios del sistema.
    *
    * Si la lista de usuarios está vacía, se mostrará en la
consola un mensaje indicando que no hay usuarios en el sistema.
```

```
    * De lo contrario, el método delega la tarea de mostrar la
lista de usuarios al método `sistema.mostrarLista`.
    */
   public void mostrarUsuarios() {
       if (sistema.getUsuarios().isEmpty()) {
           System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
           System.out.println("No hay usuarios en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
       } else {
           sistema.mostrarLista(sistema.getUsuarios(),
"usuarios");
       }
   }

   /**
    * Elimina un usuario del sistema.
    *
    * Este método verifica si existen usuarios en el sistema
antes de proceder
    * a eliminarlos. Si no hay usuarios registrados, informa que
el sistema
    * está vacío. En caso contrario, delega la eliminación al
método
    * `eliminarPorDni`, solicitando al usuario el DNI del usuario
que desea
    * eliminar. Notifica al usuario sobre el éxito o fracaso de
la operación.
    */
   public void eliminarUsuario() {
       if (sistema.getUsuarios().isEmpty()) {
           System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
           System.out.println("No hay usuarios en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
       } else {
           eliminarPorDni(sistema.getUsuarios(), u -> u.dni,
"usuarios");
       }
```

```
    }

    /**
     * Edita los datos de un usuario existente en el sistema.
     *
     * Este método comprueba si el sistema contiene usuarios antes
de continuar.
     * Si no hay usuarios presentes, muestra un mensaje
correspondiente. De lo contrario,
     * invoca un método de propósito general para gestionar el
proceso de edición para
     * los usuarios de la categoría "usuarios".
     */
    public void editarUsuario() {
        if (sistema.getUsuarios().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
            System.out.println("No hay usuarios en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
        } else {
            editarPersona(sistema.getUsuarios(), "usuarios"); //
No necesitamos editor de saldo para mecánicos
        }
    }

    /**
     * Gestiona las operaciones vehiculares, como la creación,
visualización, edición y eliminación.
     * Este método funciona como coordinador, permitiendo al
usuario realizar operaciones
     * en entidades de vehículos a través de una interfaz con
menús.
     *
     * El método invoca un mecanismo genérico de gestión de
entidades adaptado a vehículos, al
     * especificar operaciones específicas para crear vehículos,
mostrar una lista de vehículos,
     * editar los detalles de los vehículos y eliminarlos.
     *
     * Se valida la entrada del usuario y se activan los
submétodos correspondientes
     * según la opción seleccionada.
```

```
    */
   public void gestionarVehiculos() {
       gestionarEntidad("vehiculo",
               this::elegirCrearVehiculo,
               this::mostrarVehiculos,
               this::editarVehiculo,
               this::eliminarVehiculo,
               new String[]{"1", "2", "3", "4"});
   }

   /**
    * Permite al usuario seleccionar y crear un tipo específico
de vehículo entre las opciones disponibles:
    * Bicicleta, scooter o motocicleta. El usuario introduce una
opción numérica correspondiente al
    * tipo de vehículo deseado. Si el usuario introduce una
cadena vacía, el proceso finaliza.
    *
    * Este método utiliza un bucle para mostrar el menú
repetidamente hasta que el usuario decide salir.
    **/
   public void elegirCrearVehiculo() {
       String accion;
       do {
           System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
           System.out.println("Que tipo de vehículo desea crear,
introduzca el número correspondiente:" + '\n' +
                   "1- Bicicleta" + '\n' + "2- Patinete" + '\n'
+ "3- Moto" + '\n' +
                   "Para salir introduzca una cadena vacía" +
'\n');
           accion = scanner.nextLine().trim();
           if (!accion.trim().isEmpty()) {
               switch (accion) {
                   case "1":
                       crearBicicleta();
                       break;
                   case "2":
                       crearPatinete();
                       break;
                   case "3":
                       crearMoto();
                       break;
```

```
                    default:
                        System.out.println("No ha introducido una
opción correcta");
                }
            } else {
                System.out.println("Saliendo...");
            }
        } while (!accion.trim().isEmpty());

    }

    /**
     * Crea un nuevo patinete y lo registra en el sistema si es
posible.
     *
     * Este método verifica primero si el sistema cuenta con bases
registradas.
     * Si no hay bases, muestra un mensaje indicando que estas son
necesarias
     * para crear un patinete. Si hay bases disponibles, se
utiliza
     * el método genérico {@code crearVehiculoGenerico} para
gestionar
     * el proceso de creación y registro del patinete.
     */
    public void crearPatinete() {
        if (sistema.getBases().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
            System.out.println("No hay bases registradas en el
sistema y son necesarias para crear un patinete");

System.out.println("-----------------------------------------------
-----------------------------------------------" + '\n');
        }else {
            crearVehiculoGenerico("Patinete", datos -> new
Patinete(
                    (int) datos.get("id")
            ));
        }
    }

    /**
```

```
    * Método encargado de crear una nueva bicicleta en el
sistema.
    *
    * Este método verifica primero si existen bases registradas
en el sistema,
    * ya que son necesarias para asignar la bicicleta
correctamente. Si no hay
    * bases disponibles, muestra un mensaje informativo y no
realiza la creación.
    * En caso contrario, utiliza el método genérico para crear
    */
   public void crearBicicleta() {
       if (sistema.getBases().isEmpty()) {
           System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
           System.out.println("No hay bases registradas en el
sistema y son necesarias para crear una bicicleta");

System.out.println("-----------------------------------------------
-----------------------------------------------" + '\n');
       }else {
           crearVehiculoGenerico("Bicicleta", datos -> new
Bicicleta(
                   (int) datos.get("id")
           ));
       }
   }

   /**
    * Crea una nueva instancia de motocicleta y la añade a la
lista de vehículos del sistema.
    * Este método utiliza un proceso genérico de creación de
vehículos, especificando "Moto" como tipo de vehículo.
    * y proporcionando un constructor para crear instancias de
"Moto".
    *
    * Si ya existe una motocicleta con el mismo ID en el sistema,
no se añadirá ninguna nueva motocicleta.
    * Se mostrará un mensaje indicando la duplicación.
    *
    * Si el proceso de creación detecta datos no válidos o
faltantes, se mostrarán los mensajes de error correspondientes.
    * */
   public void crearMoto() {
```

```
        crearVehiculoGenerico("Moto", datos -> new Moto(
                (int) datos.get("id"),
                (int) datos.get("coordX"),
                (int) datos.get("coordY"),
                (String) datos.get("cilindrada")
        ));
    }

    /**
     * Crea un vehículo a partir del tipo especificado,
solicitando al usuario los atributos necesarios
     * según el tipo indicado. Los datos del vehículo se almacenan
en un mapa para su posterior uso.
     *
     * @param vehiculo el tipo de vehículo que se desea crear (por
ejemplo, "Moto").
     *                 Dependiendo del tipo, se pueden solicitar
atributos como coordenadas
     *                 o cilindrada específica.
     *
     * @return un mapa que contiene los atributos del vehículo. Si
ocurre un error o
     *         el vehículo no se puede crear (por ejemplo,
coordenadas fuera de los límites),
     *         se devuelve null.
     */
    public Map<String, Object> crearVehiculo(String vehiculo) {
        Map<String, Object> datos = new HashMap<>();
        try {
            System.out.println("Introduce el id de " + vehiculo);
            datos.put("id",
Integer.parseInt(scanner.nextLine()));
            if (vehiculo.equalsIgnoreCase("Moto")) {
                System.out.println("Introduce la coordenada X de
" + vehiculo + " debe estar entre 0 y " +
sistema.getLimiteX());
                datos.put("coordX",
Integer.parseInt(scanner.nextLine()));
                System.out.println("Introduce la coordenada Y de
" + vehiculo + " debe estar entre 0 y " +
sistema.getLimiteY());
                datos.put("coordY",
Integer.parseInt(scanner.nextLine()));
                System.out.println("Introduce la cilindrada de la
moto , puede ser: grande o pequeña");
```

```
                datos.put("cilindrada", scanner.nextLine());
                if ((int) datos.get("coordX") < 0 || (int)
datos.get("coordX") > sistema.getLimiteX() || (int)
datos.get("coordY") < 0 || (int) datos.get("coordY") >
sistema.getLimiteY()) {
                    System.out.println("Las coordenadas están
fuera de los límites de la ciudad");
                    return null;
                } else {
                    return datos;
                }
            }
            return datos;
        } catch (IllegalArgumentException e) {
            System.err.println("Error al crear el vehiculo: " +
e.getMessage());
        }
        return null;
    }

    /**
     * Muestra la lista de vehículos almacenados en el sistema.
     *
     * Si no hay vehículos registrados, se imprime un mensaje
indicando
     * que no hay vehículos en el sistema.
     * En caso contrario, se invoca el método {@code mostrarLista}
del sistema
     * para listar todos los vehículos disponibles con el
encabezado correspondiente.
     */
    public void mostrarVehiculos() {
        if (sistema.getVehiculos().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
----------------------------");
            System.out.println("No hay vehiculos en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------------" + '\n');
        } else {
            sistema.mostrarLista(sistema.getVehiculos(),
"vehiculos");
        }
    }
```

```java
    /**
     * Elimina un vehículo del sistema según su identificador
(ID), excepto en el caso de motocicletas (Moto).
     *
     * Este método verifica si la lista de vehículos está vacía
antes de proceder.
     * Si no hay vehículos, se informa al usuario. Si hay
vehículos, se muestra una lista de los mismos para
     * que el usuario pueda seleccionar el vehículo a eliminar
introduciendo su ID.
     *
     * En caso de que el usuario introduzca un ID que no sea un
número entero, se informa de que la entrada es inválida.
     *
     * Excepciones:
     * - NumberFormatException: Si la entrada del usuario*/
    public void eliminarVehiculo() {
        if (sistema.getVehiculos().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
            System.out.println("No hay vehiculos en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------------" + '\n');
        } else {
            sistema.mostrarLista(sistema.getVehiculos(),
"vehiculos");
            System.out.println("Introduce el id del vehiculo que
quiere eliminar");
            try {
                int id = Integer.parseInt(scanner.nextLine());
                Vehiculo eliminar = null;
                for (Vehiculo v : sistema.getVehiculos()) {
                    if (v.id == id && !(v instanceof Moto)) {
                        eliminar = v;
                    }
                }
                if (eliminar != null) {
                    if (eliminar instanceof Patinete) {
                        for (Base b : sistema.getBases()) {
                            for (Vehiculo v :
b.getPatinetesDisponibles()) {
                                if (v.id == eliminar.id) {
```

```
b.eliminarVehiculoDisponible(v);
                                }
                            }
                        }
                    }
                    if (eliminar instanceof Bicicleta) {
                        for (Base b : sistema.getBases()) {
                            for (Vehiculo v :
b.getBicicletasDisponibles()) {
                                if (v.id == eliminar.id) {

b.eliminarVehiculoDisponible(v);
                                }
                            }
                        }
                    }

                }
                boolean eliminado =
sistema.getVehiculos().removeIf(v -> v.id == id);

                if (eliminado) {
                    System.out.println("Vehiculo eliminado");
                } else {
                    System.out.println("No se ha encontrado el
vehiculo");
                }
            } catch (NumberFormatException e) {
                System.out.println('\n' +"Entrada inválida. " +
'\n');
            }
        }
    }

    /**
     * Permite editar los detalles de un vehículo en el sistema.
     *
     * El método valida si existen vehículos registrados en el
sistema antes de proceder.
     * Si no existen, muestra un mensaje indicando que no hay
vehículos en el sistema.
     * En caso de haberlos, se solicita al usuario el ID del
vehículo que desea editar.
     *
```

```
    * En caso de un error de entrada como un ID no numérico o no
existente, notifica al usuario
    * y no realiza los cambios.
    *
    * Además, el menú de edición permite al usuario salir en
cualquier*/
   public void editarVehiculo() {
       if (sistema.getVehiculos().isEmpty()) {
           System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
           System.out.println("No hay vehiculos en el sistema");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
       } else {
           sistema.mostrarLista(sistema.getVehiculos(),
"vehiculos");
           System.out.println("Introduce el ID del vehículo que
quieres editar:");
           int id;
           try {
               id = Integer.parseInt(scanner.nextLine());
           } catch (NumberFormatException e) {
               System.out.println("El ID debe ser un número
entero.");
               return;
           }

           Vehiculo vehiculoEditar =
sistema.getVehiculos().stream()
                   .filter(v -> v.id == id)
                   .findFirst()
                   .orElse(null);

           if (vehiculoEditar == null) {
               System.out.println("No se ha encontrado un
vehículo con ese ID.");
           } else {
               String opcion;
               do {
                   mostrarMenuEditarVehiculo(vehiculoEditar);
                   opcion = scanner.nextLine().trim();

                   try {
```

```
                        switch (opcion) {
                            case "1" ->
editarNivelBateria(vehiculoEditar);
                            case "2" ->
editarEstado(vehiculoEditar);
                            case "3" ->
editarCoordenada(vehiculoEditar, true);
                            case "4" ->
editarCoordenada(vehiculoEditar, false);
                            case "5" ->
editarCilindrada(vehiculoEditar);
                            case "" ->
System.out.println("Saliendo...");
                            default -> System.out.println("Opción
no válida.");
                        }
                    } catch (NumberFormatException e) {
                        System.out.println('\n' +"Entrada
inválida. Se esperaba un número." + '\n');
                    }
                } while (!opcion.isEmpty());
            }

        }
    }

    /**
     * Muestra el menú para editar las propiedades de un vehículo
específico.
     * Este menú varía dependiendo del tipo de vehículo, mostrando
opciones adicionales
     * si se trata de una motocicleta.
     *
     * @param vehiculo El vehículo que será editado, incluyendo
atributos como nivel de batería, estado,
     *                 y atributos específicos dependiendo del
tipo de vehículo (por ejemplo, coordenadas
     *                 o cilindrada en caso de una motocicleta).
     */
    private void mostrarMenuEditarVehiculo(Vehiculo vehiculo) {
        System.out.println('\n' +
"-------------------------------------------------------------
----------------------------");
        System.out.println("Vehículo encontrado: " + vehiculo);
        System.out.println("¿Qué desea editar?");
```

```
        System.out.println("1 - Nivel de batería");
        System.out.println("2 - Estado");
        if (vehiculo instanceof Moto) {
            System.out.println("3 - Coordenada X");
            System.out.println("4 - Coordenada Y");
            System.out.println("5 - Cilindrada");
        }
        System.out.println("Pulse Enter para salir");

System.out.println("-----------------------------------------------
-----------------------------------------");
    }

    /**
     * Actualiza el nivel de batería de un vehículo y sincroniza
su información
     * en la base correspondiente si pertenece a una lista de
patinetes o bicicletas disponibles.
     *
     * @param vehiculo Objeto de la clase Vehiculo cuyo nivel de
batería se va a editar.
     */
    private void editarNivelBateria(Vehiculo vehiculo) {
        System.out.println("Nuevo nivel de batería:");
        double nivel = 100;
        try {
            nivel = Double.parseDouble(scanner.nextLine());
        } catch (IllegalArgumentException e) {
            System.out.println("Error al añadir el saldo: " +
e.getMessage());
        }
        vehiculo.setNivelBateria(nivel);
        if (vehiculo instanceof Patinete) {
            for (Base b : sistema.getBases()) {
                for (Vehiculo v : b.getPatinetesDisponibles()) {
                    if (v.id == vehiculo.id) {
                        b.eliminarVehiculoDisponible(v);
                        b.agregarVehiculoDisponible(vehiculo);
                    }
                }
            }
        }
        if (vehiculo instanceof Bicicleta) {
            for (Base b : sistema.getBases()) {
```

```
                for (Vehiculo v : b.getBicicletasDisponibles())
{
                    if (v.id == vehiculo.id) {
                        b.eliminarVehiculoDisponible(v);
                        b.agregarVehiculoDisponible(vehiculo);
                    }
                }
            }
        }
        System.out.println("Nivel de batería actualizado.");
    }

    /**
     * Actualiza el estado de un vehículo y realiza los cambios
necesarios en las bases
     * correspondientes para reflejar el nuevo estado.
     *
     * @param vehiculo El vehículo cuyo estado se desea editar.
Puede ser una bicicleta,
     *                 un patinete, o cualquier subclase de
Vehiculo.
     */
    private void editarEstado(Vehiculo vehiculo) {
        System.out.println("Nuevo estado (disponible, reservado o
averiado):");
        String estado = scanner.nextLine().trim().toLowerCase();
        try {
            vehiculo.setEstado(estado);
            if (vehiculo instanceof Patinete) {
                for (Base b : sistema.getBases()) {
                    for (Vehiculo v :
b.getPatinetesDisponibles()) {
                        if (v.id == vehiculo.id) {
                            b.eliminarVehiculoDisponible(v);

b.agregarVehiculoDisponible(vehiculo);
                        }
                    }
                }
            }
            if (vehiculo instanceof Bicicleta) {
                for (Base b : sistema.getBases()) {
                    for (Vehiculo v :
b.getBicicletasDisponibles()) {
                        if (v.id == vehiculo.id) {
```

```
                            b.eliminarVehiculoDisponible(v);

b.agregarVehiculoDisponible(vehiculo);
                        }
                    }
                }
            }
            System.out.println("Estado actualizado.");
        } catch (IllegalArgumentException e) {
            System.err.println(e);
        }

    }

    /**
     * Actualiza la coordenada X o Y de un objeto de tipo Moto si
el vehículo proporcionado es una instancia de Moto.
     * Si el vehículo no es del tipo Moto, muestra un mensaje
indicando que la operación no es válida.
     * Solicita al usuario un valor numérico para la nueva
coordenada.
     *
     * @param vehiculo*/
    private void editarCoordenada(Vehiculo vehiculo, boolean esX)
{
        if (vehiculo instanceof Moto moto) {
            System.out.println("Nueva coordenada " + (esX ? "X" :
"Y") + ":");
            try {
                int valor = Integer.parseInt(scanner.nextLine());
                if (esX) {
                    moto.setCoordX(valor);
                    System.out.println("Coordenada X
actualizada.");
                } else {
                    moto.setCoordY(valor);
                    System.out.println("Coordenada Y
actualizada.");
                }
            } catch (NumberFormatException e) {
                System.out.println("El ID debe ser un número
entero.");
                return;
            }
        } else {
```

```
            System.out.println("Opción no válida para este tipo
de vehículo.");
        }
    }

    /**
     * Permite editar la cilindrada de un vehículo en caso de que
este sea una moto.
     * La cilindrada puede establecerse como "grande" o "pequeña".
Si el vehículo no
     * es una moto, se notifica que la opción no es válida para
ese tipo de vehículo.
     *
     * @param vehiculo El objeto de tipo Vehiculo cuya cilindrada
se desea actualizar.
     */
    private void editarCilindrada(Vehiculo vehiculo) {
        if (vehiculo instanceof Moto moto) {
            try {
                System.out.println("Nueva cilindrada (grande o
pequeña):");
                String cilindrada =
scanner.nextLine().trim().toLowerCase();
                moto.setCilindrada(cilindrada);
                System.out.println("Cilindrada actualizada.");
            } catch (IllegalArgumentException e) {
                System.err.println(e);
            }
        } else {
            System.out.println("Opción no válida para este tipo
de vehículo.");
        }
    }

    /**
     * Permite al usuario gestionar las tarifas de varios tipos de
vehículos mediante un menú en la consola.
     * El método ofrece opciones para modificar, ver o salir del
proceso de gestión de tarifas.
     */
    public void gestionarTarifas() {
        String accion = "1";
        while (!accion.trim().isEmpty()) {
```

```
            System.out.println('\n' +
"-------------------------------------------------------------------
--------------------------" + '\n');
            System.out.println("La tarifa de que vehiculo quiere
modificar" + '\n' +
                    "1- Moto" + '\n' + "2- Bicicleta" + '\n' +
"3- Patinete" + '\n' +
                    "4- Ver tarifas" + '\n' +
                    "Pulse Enter para salir" + '\n');
            accion = scanner.nextLine().trim();
            if (!accion.trim().isEmpty()) {
                if (accion.equalsIgnoreCase("1")) {
                    editarTarifas(TipoVehiculos.MOTO);
                } else if (accion.equalsIgnoreCase("2")) {
                    editarTarifas(TipoVehiculos.BICICLETA);
                } else if (accion.equalsIgnoreCase("3")) {
                    editarTarifas(TipoVehiculos.PATINETE);
                } else if (accion.equalsIgnoreCase("4")) {
                    mostrarTarifas();
                } else {
                    System.out.println("No ha introducido una
opcion correcta");
                }
            } else {
                System.out.println("Saliendo...");
            }
        }
    }

    /**
     * Permite visualizar y editar las tarifas asociadas a un tipo
específico de vehículo.
     * Ofrece opciones para visualizar las tarifas actuales,
editar el precio por minuto
     * o modificar el descuento premium de un tipo de vehículo
específico.
     *
     * @param tipo El tipo de vehículo para el que se desea
gestionar las tarifas. Puede ser
     *             BICICLETA, PATINETE o MOTO.
     */
    public void editarTarifas(TipoVehiculos tipo) {
        String accion = "1";
        while (!accion.trim().isEmpty()) {
```

```
            System.out.println('\n' +
"-------------------------------------------------------------------
--------------------------" + '\n');
            System.out.println("Que desea hacer" + '\n' +
                    "1- Ver la tarifa de " + tipo + '\n' + "2-
Editar el precio por minuto" + '\n' + "3- Editar el descuento
premium" + '\n' +
                    "Pulse Enter para salir" + '\n');
            accion = scanner.nextLine().trim();
            if (!accion.trim().isEmpty()) {
                if (accion.equalsIgnoreCase("1")) {
                    mostrarTarifa(tipo);
                } else if (accion.equalsIgnoreCase("2")) {
                    editarPrecioMinuto(tipo);
                } else if (accion.equalsIgnoreCase("3")) {
                    editarDescuentoPremium(tipo);
                } else {
                    System.out.println("No ha introducido una
opcion correcta");
                }
            } else {
                System.out.println("Saliendo...");
            }
        }
    }

    /**
     * Muestra la información de tarifas para un tipo de vehículo
específico.
     *
     * @param tipo: el tipo de vehículo para el que se mostrará la
tarifa.
     * Puede ser: moto, bicicleta o patinete.
     */
    public void mostrarTarifa(TipoVehiculos tipo) {
        if (tipo == TipoVehiculos.BICICLETA) {

System.out.println(sistema.tarifaBicicleta.toString());
        } else if (tipo == TipoVehiculos.PATINETE) {

System.out.println(sistema.tarifaPatinete.toString());
        } else if (tipo == TipoVehiculos.MOTO) {
            System.out.println(sistema.tarifaMoto.toString());
        }
    }
```

```java
   /**
    * Modifica el precio por minuto de alquiler de un tipo
específico de vehículo.
    *
    * @param tipo el tipo de vehículo cuyo precio por minuto se
desea editar.
    *             Puede ser uno de los valores de la enumeración
TipoVehiculos
    *             como BICICLETA, PATINETE o MOTO.
    */
   public void editarPrecioMinuto(TipoVehiculos tipo) {

       String tipoVehiculo = "";
       int descuento = 0;
       int precio = 0;
       if (tipo == TipoVehiculos.BICICLETA) {
           precio =
sistema.tarifaBicicleta.getPrecioPorMinuto();
           precio = mostrarPrecioMinuto("bicicleta", precio);
           if (precio < 0) {
               System.out.println("El precio debe ser
positivo");
           } else {

sistema.tarifaBicicleta.setPrecioPorMinuto(precio);
               System.out.println('\n' + "Precio editado
correctamente");
           }
       } else if (tipo == TipoVehiculos.PATINETE) {
           precio =
sistema.tarifaPatinete.getPrecioPorMinuto();
           precio = mostrarPrecioMinuto("patinete", precio);
           if (precio < 0) {
               System.out.println("El precio debe ser
positivo");
           } else {

sistema.tarifaPatinete.setPrecioPorMinuto(precio);
               System.out.println('\n' + "Precio editado
correctamente");
           }
       } else if (tipo == TipoVehiculos.MOTO) {
           precio = sistema.tarifaMoto.getPrecioPorMinuto();
           precio = mostrarPrecioMinuto("moto", precio);
```

```
            if (precio < 0) {
                System.out.println("El precio debe ser
positivo");
            } else {
                sistema.tarifaMoto.setPrecioPorMinuto(precio);
                System.out.println('\n' + "Precio editado
correctamente");
            }
        }
    }

    /**
     * Muestra el precio actual por minuto para un tipo de
vehículo específico.
     * Permite al usuario introducir un nuevo precio y actualizar
el valor si la entrada es válida.
     *
     * @param tipoVehiculo El tipo de vehículo para el que se
muestra y actualiza el precio por minuto.
     * @param precio El precio actual por minuto para el tipo de
vehículo especificado.
     * @return El precio por minuto actualizado si se proporciona
una entrada válida; de lo contrario, devuelve -1
     * si la entrada es inválida.
     */
    public int mostrarPrecioMinuto(String tipoVehiculo, int
precio) {
        System.out.println('\n' + "El precio por minuto actual de
" + tipoVehiculo + " es de: " + precio + "€" + '\n');
        System.out.println("Introduzca el nuevo precio por
minuto, solo se aceptan unidades");
        try {
            precio = Integer.parseInt(scanner.nextLine());
        } catch (Exception e) {
            System.out.println('\n' +"Entrada inválida. " +
'\n');
            precio = -1;
        }

        return precio;
    }

    /**
     * Modifica el descuento premium asociado a un tipo de
vehículo específico
```

```
    * (bicicleta, patinete o moto). Permite asignar un nuevo
valor al descuento
    * premium, verificando que este se encuentre en un rango
válido entre 0 y 100.
    * Si el valor ingresado no cumple con las restricciones, se
notifica al usuario.
    *
    * @param tipo el tipo de vehículo cuyo descuento premium se
desea editar.
    *             Puede ser TipoVehiculos.BICICLETA,
TipoVehiculos.PATINETE o
    *             TipoVehiculos.MOTO.
    */
   public void editarDescuentoPremium(TipoVehiculos tipo) {
       String tipoVehiculo = "";
       int descuento = 0;
       int descuentoPremium = 0;
       if (tipo == TipoVehiculos.BICICLETA) {
           descuentoPremium =
sistema.tarifaBicicleta.getDescuentoPremium();
           descuentoPremium =
mostrarDescuentoPremium("bicicleta", descuentoPremium);
           if (descuentoPremium < 0 || descuentoPremium > 100)
{
               System.out.println("El descuento premium debe
estar entre 0 y 100");
           } else {

sistema.tarifaBicicleta.setDescuentoPremium(descuentoPremium);
               System.out.println('\n' + "Descuento editado
correctamente");
           }
       } else if (tipo == TipoVehiculos.PATINETE) {
           descuentoPremium =
sistema.tarifaPatinete.getDescuentoPremium();
           descuentoPremium =
mostrarDescuentoPremium("patinete", descuentoPremium);
           if (descuentoPremium < 0 || descuentoPremium > 100)
{
               System.out.println("El descuento premium debe
estar entre 0 y 100");
           } else {

sistema.tarifaPatinete.setDescuentoPremium(descuentoPremium);
```

```
                System.out.println('\n' + "Descuento editado
correctamente");
            }
        } else if (tipo == TipoVehiculos.MOTO) {
            descuentoPremium =
sistema.tarifaMoto.getDescuentoPremium();
            descuentoPremium = mostrarDescuentoPremium("moto",
descuentoPremium);
            if (descuentoPremium < 0 || descuentoPremium > 100)
{
                System.out.println("El descuento premium debe
estar entre 0 y 100");
            } else {

sistema.tarifaMoto.setDescuentoPremium(descuentoPremium);
                System.out.println('\n' + "Descuento editado
correctamente");
            }
        }
    }

    /**
     * Muestra el descuento premium actual para el tipo de
vehículo especificado y permite al usuario actualizarlo.
     * Si el valor introducido no es válido, el descuento se
mantiene y se muestra un mensaje de error.
     *
     * @param tipoVehiculo: el tipo de vehículo para el que se
muestra y modifica el descuento premium.
     * @param descuentoPremium: el valor actual del descuento
premium para el tipo de vehículo especificado.
     * @return: el valor actualizado del descuento premium o -1 si
se proporcionó una entrada no válida.
     */
    public int mostrarDescuentoPremium(String tipoVehiculo, int
descuentoPremium) {
        System.out.println('\n' + "El descuento premium actual de
" + tipoVehiculo + " es de: " + descuentoPremium + "%" + '\n');
        System.out.println("Introduzca el nuevo descuento
premium, solo se aceptan unidades");
        try {
            descuentoPremium =
Integer.parseInt(scanner.nextLine());
        } catch (Exception e) {
```

```
            System.out.println('\n' +"Entrada inválida. " +
'\n');
            descuentoPremium = -1;
        }

        return descuentoPremium;
    }

    public void mostrarTarifas() {
        System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
        System.out.println(sistema.tarifaBicicleta.toString());
        System.out.println(sistema.tarifaMoto.toString());
        System.out.println(sistema.tarifaPatinete.toString());
    }

    /**
     * Gestiona las operaciones del ciclo de vida de las mecánicas
dentro de la aplicación.
     * Este método es un controlador de alto nivel para realizar
diversas acciones, como crear,
     * mostrar, editar y eliminar entidades mecánicas.
     */
    public void gestionarMecanicos() {
        gestionarEntidad("mecánico",
                this::crearMecanico,
                this::mostrarMecanicos,
                this::editarMecanico,
                this::eliminarMecanico,
                new String[]{"1", "2", "3", "4"});
    }

    /**
     * Crea un nuevo mecánico y lo añade al sistema si no existe
otro mecánico con el mismo DNI.
     * Este método utiliza el método `crearPersona` para solicitar
los datos del mecánico y luego
     * realiza una validación con los mecánicos existentes en el
sistema según su DNI.
     **/
    public void crearMecanico() {
        crearPersona("mecánico", (dni, nombre, fechaNac) -> {
            Mecanico mecanico = new Mecanico(dni, nombre,
fechaNac);
```

```
            boolean agregado = false;
            for (Mecanico m : sistema.getMecanicos()) {
                if
(m.getDni().equalsIgnoreCase(mecanico.getDni())) {
                    agregado = true;
                }
            }
            if (!agregado) {
                sistema.getMecanicos().add(mecanico);
                System.out.println("Mecanico agregado");
            } else {
                System.out.println("Ya existe un mecanico con ese
dni, el mecanico no se ha podido agregar");
            }
        });
    }

    /**
     * Muestra la lista de mecánicas en el sistema.
     *
     * Si el sistema no tiene mecánicas, se mostrará en la consola
un mensaje indicando que no hay mecánicas.
     * */
    public void mostrarMecanicos() {
        if (sistema.getMecanicos().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
----------------------------");
            System.out.println("No hay mecánicos en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------------" + '\n');
        } else {
            sistema.mostrarLista(sistema.getMecanicos(),
"mecánicos");
        }
    }

    /**
     * Elimina a un mecánico del sistema según su DNI (Documento
Nacional de Identidad).
     * Si no hay ningún mecánico registrado en el sistema, se
muestra el mensaje correspondiente.
     * El método primero comprueba si la lista de mecánicos está
vacía.
```

```
    **/
   public void eliminarMecanico() {
       if (sistema.getMecanicos().isEmpty()) {
           System.out.println('\n' +
"-------------------------------------------------------------------
---------------------------");
           System.out.println("No hay mecánicos en el sistema");

System.out.println("-------------------------------------------------
-----------------------------------------------" + '\n');
       } else {
           eliminarPorDni(sistema.getMecanicos(), m -> m.dni,
"mecánico");
       }
   }

   /**
    * Edita los datos de un mecánico en el sistema.
    *
    * Este método comprueba si hay mecánicos registrados en el
sistema.
    * Si la lista de mecánicos del sistema está vacía, se muestra
un mensaje indicando que no hay mecánicos.
    **/
   public void editarMecanico() {
       if (sistema.getMecanicos().isEmpty()) {
           System.out.println('\n' +
"-------------------------------------------------------------------
---------------------------");
           System.out.println("No hay mecánicos en el sistema");

System.out.println("-------------------------------------------------
-----------------------------------------------" + '\n');
       } else {
           editarPersona(sistema.getMecanicos(), "mecánico");
       }
   }

   /**
    * Asigna un mecánico a un vehículo o base averiado de la
lista disponible en el sistema.
    *
    * Este método busca vehículos y bases que necesiten
reparación mecánica
```

```
    * y verifica que haya mecánicos disponibles en el sistema. Si
se cumplen ambas condiciones
    *, ofrece al usuario la opción de seleccionar un vehículo o
una base
    * para asignar un mecánico. El usuario puede seleccionar un
vehículo o una base específicos por su ID y luego
    * asignarlo a un mecánico ingresando su DNI.
    */
   public void asignarMecanico() {
       ArrayList<Vehiculo> vehiculosNecesitanMecanico =
getVehiculosAveriados();
       ArrayList<Base> basesNecesitanMecanico =
getBasesAveriadas();
       if (vehiculosNecesitanMecanico.isEmpty() &&
basesNecesitanMecanico.isEmpty()) {
           System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
           System.out.println("No hay vehículos o bases
averiadas");
           System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
       } else {
           if (sistema.getMecanicos().isEmpty()) {
               System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
               System.out.println("No hay mecanicos en el
sistema");
               System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
           } else {
               System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
               String accion = "8";
               while (!accion.trim().isEmpty()) {
                   System.out.println("Que desea reaparar?" +
'\n' +
                           "1- Bases " + '\n' +
                           "2- Vehiculos " + '\n' +
                           "Pulse Enter para salir" + '\n');
                   accion = scanner.nextLine().trim();
```

```
                if (!accion.trim().isEmpty()) {
                    switch (accion) {
                        case "1":
                            if
(basesNecesitanMecanico.isEmpty()) {
                                System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
                                System.out.println("No hay
ninguna base que necesite mecanico" + '\n');
                            } else {
                                System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');

sistema.mostrarLista(basesNecesitanMecanico, "bases que
necesitan mecanico");
                                Base baseSeleccionada =
null;
                                while (baseSeleccionada ==
null) {

System.out.println("Seleccione el id de la base que necesita
mecanico");
                                    int id = -1;
                                    try {
                                        id =
Integer.parseInt(scanner.nextLine());
                                    } catch (Exception e) {

System.out.println('\n' +"Entrada inválida. " + '\n');
                                        break;
                                    }

                                    for (Base b :
basesNecesitanMecanico) {
                                        if (b.getId() == id)
{
                                            baseSeleccionada
= b;
                                            break;
                                        }
                                    }
                                    if (baseSeleccionada ==
null) {
```

```
System.out.println('\n' +"No se ha encontrado una base con ese
ID."+ '\n');
                                        } else {

sistema.mostrarLista(sistema.getMecanicos(), "mecanicos");
                                            Mecanico
mecanicoSeleccionado = null;
                                            String
opcionMecanico;
                                            if
(mecanicoSeleccionado == null) {
                                                do {

System.out.println("Introduzca el dni del mecanico al que desea
asignar esta base");

opcionMecanico = scanner.nextLine();
                                                    if
(!opcionMecanico.trim().isEmpty()) {
                                                        for
(Mecanico m : sistema.getMecanicos()) {
                                                            if
(m.getDni().equalsIgnoreCase(opcionMecanico)) {

mecanicoSeleccionado = m;
                                                            }
                                                        }
                                                        if
(mecanicoSeleccionado == null) {

System.out.println('\n' + "No se ha encontrado mecanico con ese
DNI." + '\n');
                                                        } else {

mecanicoSeleccionado.agregaBaseAsignada(baseSeleccionada);

System.out.println('\n' +"La " + baseSeleccionada + '\n' + "ha
sido asignada a el mecanico " +
mecanicoSeleccionado.getNombre() + '\n');

accion = "";
                                                        }
                                                    }
```

```
                                                } while
(mecanicoSeleccionado == null &&
!opcionMecanico.trim().isEmpty());
                                            }
                                        }
                                    }
                                }
                                break;
                            case "2":
                                if
(vehiculosNecesitanMecanico.isEmpty()) {
                                    System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
                                    System.out.println("No hay
ningun vehiculo que necesite mecanico" + '\n');
                                } else {
                                    System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');

sistema.mostrarLista(vehiculosNecesitanMecanico, "vehiculos que
necesitan mecanico");
                                    Vehiculo
vehiculoSeleccionado = null;
                                    while (vehiculoSeleccionado
== null) {

System.out.println("Seleccione el id del vehiculo que necesita
mecanico");
                                        int id = -1;
                                        try {
                                            id =
Integer.parseInt(scanner.nextLine());
                                        } catch (Exception e) {

System.out.println('\n' +"Entrada inválida. " + '\n');
                                            break;
                                        }
                                        for (Vehiculo v :
vehiculosNecesitanMecanico) {
                                            if (v.getId() == id)
{

vehiculoSeleccionado = v;
```

```
                                                    break;
                                                }
                                            }
                                            if (vehiculoSeleccionado
== null) {

System.out.println("No se ha encontrado un vehiculo con ese
ID.");
                                            } else {

sistema.mostrarLista(sistema.getMecanicos(), "mecanicos");
                                                Mecanico
mecanicoSeleccionado = null;
                                                String
opcionMecanico;
                                                if
(mecanicoSeleccionado == null) {
                                                    do {

System.out.println("Introduzca el dni del mecanico al que desea
asignar este vehiculo");

opcionMecanico = scanner.nextLine();
                                                        if
(!opcionMecanico.trim().isEmpty()) {
                                                            for
(Mecanico m : sistema.getMecanicos()) {
                                                                if
(m.getDni().equalsIgnoreCase(opcionMecanico)) {

mecanicoSeleccionado = m;
                                                                }
                                                            }
                                                            if
(mecanicoSeleccionado == null) {

System.out.println("No se ha encontrado mecanico con ese DNI.");
                                                            } else {

mecanicoSeleccionado.agregarVehiculoAsignado(vehiculoSelecciona
do);

System.out.println('\n' + "Vehiculo " + vehiculoSeleccionado +
"asignado a mecanico"+ '\n');
                                                            }
```

```
                                                    }
                                                } while
(mecanicoSeleccionado == null &&
!opcionMecanico.trim().isEmpty());
                                            }
                                        }
                                    }
                                }
                                break;
                            default:
                                System.out.println("No ha
introducido una opción correcta");
                        }
                    } else {
                        System.out.println("Saliendo...");
                    }
                }
            }
        }

    /**
     * Gestiona los procesos de mantenimiento delegando
operaciones específicas a otros métodos.
     * Este método gestiona el ciclo de vida de las entidades de
mantenimiento, incluyendo su creación,
     * visualización, edición y eliminación.
     *
     * El método utiliza un sistema genérico de gestión de
entidades.
     * */
    public void gestionarMantenimientos() {
        gestionarEntidad("mantenimiento",
                this::crearMantenimiento,
                this::mostrarMantenimientos,
                this::editarMantenimiento,
                this::eliminarMantenimiento,
                new String[]{"1", "2", "3", "4"});
    }

    public void crearMantenimiento() {
        crearPersona("mantenimiento", (dni, nombre, fechaNac) ->
{
            Mantenimiento mantenimiento = new Mantenimiento(dni,
nombre, fechaNac);
```

```
            boolean agregado = false;
            for (Mantenimiento m : sistema.getMantenimientos())
{
                if
(m.getDni().equalsIgnoreCase(mantenimiento.getDni())) {
                    agregado = true;
                }
            }
            if (!agregado) {
                sistema.getMantenimientos().add(mantenimiento);
                System.out.println("Empleado de mantenimiento
agregado");
            } else {
                System.out.println("Ya existe un empleado de
mantenimiento con ese dni, el empleado de mantenimineto no se
ha podido agregar");
            }
        });
    }

    public void mostrarMantenimientos() {
        if (sistema.getMantenimientos().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
----------------------------");
            System.out.println("No hay empleados de mantenimiento
en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------------" + '\n');
        } else {
            sistema.mostrarLista(sistema.getMantenimientos(),
"mantenimientos");
        }
    }

    public void eliminarMantenimiento() {
        if (sistema.getMantenimientos().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
----------------------------");
            System.out.println("No hay empleados de mantenimiento
en el sistema");
```

```
System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
        } else {
            eliminarPorDni(sistema.getMantenimientos(), m ->
m.dni, "mantenimiento");
        }
    }

    public void editarMantenimiento() {
        if (sistema.getMantenimientos().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------
---------------------------");
            System.out.println("No hay empleados de mantenimiento
en el sistema");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
        } else {
            editarPersona(sistema.getMantenimientos(),
"mantenimiento");
        }
    }

    public void asignarMantenimiento() {
        ArrayList<Vehiculo> vehiculosNecesitanMantenimiento =
getVehiculosAveriados();
        for (Vehiculo v : sistema.getVehiculos()) {
            if (v.getNivelBateria() < 20 && v.getEstado() ==
EstadosVehiculo.DISPONIBLE) {
                vehiculosNecesitanMantenimiento.add(v);
            }
        }
        if (vehiculosNecesitanMantenimiento.isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------
---------------------------" + '\n');
            System.out.println("No hay ningun vehiculo que
necesite mantenimento");
            System.out.println('\n' +
"-----------------------------------------------------------
---------------------------" + '\n');
        } else {
            if (sistema.getMantenimientos().isEmpty()) {
```

```
                System.out.println('\n' +
"-------------------------------------------------------------
---------------------------" + '\n');
                System.out.println("No hay empleados de
mantenimiento en el sistema");
                System.out.println('\n' +
"-------------------------------------------------------------
---------------------------" + '\n');
            } else {
                System.out.println('\n' +
"-------------------------------------------------------------
---------------------------" + '\n');

sistema.mostrarLista(vehiculosNecesitanMantenimiento,
"vehiculos que necesitan mantenimiento");
                Vehiculo vehiculoSeleccionado = null;
                String opcion = ".";
                while (vehiculoSeleccionado == null &&
!opcion.trim().isEmpty()) {
                    System.out.println("Seleccione el id del
vehiculo que necesita mantenimiento");
                    int id = -1;
                    try {
                        id =
Integer.parseInt(scanner.nextLine());
                    } catch (Exception e) {
                        System.out.println('\n' +"Entrada
inválida. " + '\n');
                        break;
                    }
                    for (Vehiculo v :
vehiculosNecesitanMantenimiento) {
                        if (v.id == id) {
                            vehiculoSeleccionado = v;
                            break;
                        }
                    }
                    if (vehiculoSeleccionado == null) {
                        System.out.println('\n' +"No se ha
encontrado un vehiculo con ese ID." + '\n');
                    } else {

sistema.mostrarLista(sistema.getMantenimientos(), "trabajadores
de mantenimiento");
```

```
                        Mantenimiento mantenimientoSeleccionado
= null;
                        String opcionMantenimiento;
                        if (mantenimientoSeleccionado == null) {
                            do {
                                System.out.println("Introduzca el
dni del trabajador de mantenimiento al que desea asignar este
vehiculo");
                                opcionMantenimiento =
scanner.nextLine();
                                if
(!opcionMantenimiento.trim().isEmpty()) {
                                    for (Mantenimiento m :
sistema.getMantenimientos()) {
                                        if
(m.getDni().equalsIgnoreCase(opcionMantenimiento)) {

mantenimientoSeleccionado = m;
                                        }
                                    }
                                    if
(mantenimientoSeleccionado == null) {
                                        System.out.println('\n' +
"No se ha encontrado un trabajador de mantenimiento con ese
DNI." + '\n');
                                    } else {
                                        if
(vehiculoSeleccionado.getEstado().toString().equalsIgnoreCase("
disponible")) {

sistema.getVehiculos().remove(vehiculoSeleccionado);

vehiculoSeleccionado.setEstado("reservado");

sistema.getVehiculos().add(vehiculoSeleccionado);
                                        }

mantenimientoSeleccionado.agregarVehiculoAsignado(vehiculoSelec
cionado);
                                        System.out.println('\n' +
"Vehiculo " + vehiculoSeleccionado + "asignado al trabajador de
mantenimiento" + '\n');
                                    }
                                }
```

```
                        } while (mantenimientoSeleccionado
== null && !opcionMantenimiento.trim().isEmpty());
                    }
                }
            }
        }
    }

    public void gestionarBases() {
        gestionarEntidad("Base",
                this::crearBase,
                this::mostrarBases,
                this::editarBase,
                this::eliminarBase,
                new String[]{"1", "2", "3", "4"});
    }

    public void crearBase() {
        Base base = null;
        try {
            System.out.println("Introduce el id de la base");
            int id = Integer.parseInt(scanner.nextLine());
            for (Base b : sistema.getBases()) {
                if (b.getId() == id) {
                    base = b;
                }
            }
            if (base == null) {
                System.out.println("Introduce la capacidad de la
base");
                int capacidad =
Integer.parseInt(scanner.nextLine());
                if (capacidad < 0) {
                    System.out.println("La capacidad de la base
debe ser un número entero positivo");
                } else {
                    System.out.println("Introduce la coordenada X
de la base");
                    int coordX =
Integer.parseInt(scanner.nextLine());
                    System.out.println("Introduce la coordenada Y
de la base");
                    int coordY =
Integer.parseInt(scanner.nextLine());
```

```
                    if (coordX < 0 || coordX >
sistema.getLimiteX() || coordY < 0 || coordY >
sistema.getLimiteY()) {
                        System.out.println("Las coordenadas están
fuera de los límites de la ciudad");
                    } else {
                        base = new Base(id, capacidad, coordX,
coordY);
                        sistema.getBases().add(base);
                        System.out.println("Base agregada");
                    }
                }
            } else {
                System.out.println("La base con id " + id + " ya
existe");
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Error al crear la base: " +
e.getMessage());
        }
    }

    public void mostrarBases() {
        if (sistema.getBases().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
            System.out.println("No hay bases en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
        } else {
            sistema.mostrarLista(sistema.getBases(), "bases");
        }
    }

    public void eliminarBase() {
        if (sistema.getBases().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
            System.out.println("No hay bases en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
```

```
        } else {
            eliminarPorId(sistema.getBases(), b -> b.id,
"bases");
        }
    }

    public void editarBase() {
        if (sistema.getBases().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
            System.out.println("No hay bases en el sistema");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
        } else {
            sistema.mostrarLista(sistema.getBases(), "bases");
            System.out.println("Introduce el id de la base que
quieres editar");
            int id = -1;
            try {
                id = Integer.parseInt(scanner.nextLine());
            } catch (Exception e) {
                System.out.println('\n' +"Entrada inválida. " +
'\n');
            }

            Base baseEditar = null;

            for (Base b : sistema.getBases()) {
                if (b.id == id) {
                    baseEditar = b;
                    break;
                }
            }

            if (baseEditar == null) {
                System.out.println("No se ha encontrado una base
con ese ID.");
                return;
            }

            String opcion = "a";
            while (!opcion.trim().isEmpty()) {
```

```
                System.out.println('\n' +
"-------------------------------------------------------------
---------------------------" + '\n');
                System.out.println("Base encontrada: " +
baseEditar);
                System.out.println("¿Qué desea editar?" + '\n' +
                        "1 - Capacidad" + '\n' +
                        "2 - Coordenada X" + '\n' +
                        "3 - Coordenada Y" + '\n' +
                        "Pulse Enter para salir");
                opcion = scanner.nextLine();

                try {
                    if (opcion.equals("1")) {
                        System.out.println("Nueva capacidad:");
                        int nuevaCapacidad =
Integer.parseInt(scanner.nextLine());
                        if (nuevaCapacidad < 0) {
                            System.out.println("La capacidad de
la base debe ser un número entero positivo");
                        } else {
                            if (nuevaCapacidad <
baseEditar.getVehiculosDisponibles().size()) {
                                System.out.println("No se puede
reducir tanto la capacidad de la base, ya que hay " +
baseEditar.getVehiculosDisponibles().size() + " vehículos
disponibles");
                            } else {

baseEditar.setCapacidad(nuevaCapacidad);
                                System.out.println("Capacidad
actualizada.");
                            }
                        }
                    } else if (opcion.equals("2")) {
                        System.out.println("Nueva coordenada
X:");
                        int nuevaX =
Integer.parseInt(scanner.nextLine());
                        if (nuevaX < 0 || nuevaX >
sistema.getLimiteX()) {
                            System.out.println("Las coordenadas
están fuera de los límites de la ciudad");
                        } else {
                            baseEditar.setCoordX(nuevaX);
```

```
                            System.out.println("Coordenada X
actualizada.");
                        }
                    } else if (opcion.equals("3")) {
                        System.out.println("Nueva coordenada
Y:");
                        int nuevaY =
Integer.parseInt(scanner.nextLine());
                        if (nuevaY < 0 || nuevaY >
sistema.getLimiteY()) {
                            System.out.println("Las coordenadas
están fuera de los límites de la ciudad");
                        } else {
                            baseEditar.setCoordY(nuevaY);
                            System.out.println("Coordenada Y
actualizada.");
                        }
                    } else if (!opcion.trim().isEmpty()) {
                        System.out.println("Opción no válida.");
                    } else {
                        System.out.println("Saliendo...");
                    }
                } catch (NumberFormatException e) {
                    System.out.println('\n' +"Entrada inválida. "
+ '\n');
                }
            }
        }
    }

    public ArrayList<Base> getBasesAveriadas() {
        ArrayList<Base> basesAveriadas = new ArrayList<>();
        for (Base b : sistema.getBases()) {
            if (b.getTieneFallosMecanicos()) {
                basesAveriadas.add(b);
            }
        }
        return basesAveriadas;
    }

    public void agregarABase(Vehiculo vehiculo) {
        Base base = null;
        while (base == null) {
```

```
sistema.mostrarLista(sistema.getBasesDisponiblesConHueco(),
"bases disponibles");
            System.out.println("Introduce el id de la base en la
que quieres añadir el vehiculo: ");
            int id = -1;
            try {
                id = Integer.parseInt(scanner.nextLine());
            } catch (Exception e) {
                System.out.println('\n' +"Entrada inválida. " +
'\n');
            }
            for (Base b : sistema.getBasesDisponiblesConHueco())
{
                if (b.getId() == id) {
                    base = b;
                }
            }
            if (base != null) {
                base.agregarVehiculoDisponible(vehiculo);
                System.out.println("Vehiculo agregado.");
            } else {
                System.out.println("No se ha encontrado una base
con ese id");
            }
        }
    }

    public ArrayList<Vehiculo> getVehiculosAveriados() {
        ArrayList<Vehiculo> vehiculosAveriados = new
ArrayList();
        ArrayList<Vehiculo> vehiculosEnReparacion = new
ArrayList();
        for (Mantenimiento m : sistema.getMantenimientos()) {
            for (Vehiculo v : m.getVehiculosAsignados()) {
                vehiculosEnReparacion.add(v);
            }
        }
        for (Mecanico m : sistema.getMecanicos()) {
            for (Vehiculo v : m.getVehiculosAsignados()) {
                vehiculosEnReparacion.add(v);
            }
        }
        for (Vehiculo v : sistema.getVehiculos()) {
            if (v.getEstado() == EstadosVehiculo.AVERIADO) {
```

```
                if (!vehiculosEnReparacion.contains(v)) {
                    vehiculosAveriados.add(v);
                }
            }
        }
        return vehiculosAveriados;
    }

    public void promocionarUsuarioPremium() {
        listadoUsuariosPremium();
        if (sistema.getPosiblesUsuariosPremium().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
            System.out.println("No hay usuarios premium que
puedan ser promocionados");
            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------" + '\n');
        } else {
            mostrarPosiblesUsuaiosPremium();
            Usuario usuario = null;
            String dni = ".";
            while (usuario == null && !dni.trim().isEmpty()) {
                System.out.println("Introduce el dni del usuario
que quieres promover a premium: ");
                dni = scanner.nextLine();
                for (Usuario u :
sistema.getPosiblesUsuariosPremium()) {
                    if (u.getDni().equals(dni)) {
                        usuario = u;
                    }
                }
                if (usuario != null) {

sistema.getPosiblesUsuariosPremium().remove(usuario);
                    Usuario usuarioPremium = new
UsuarioPremium(usuario.getDni(), usuario.getNombre(),
usuario.getfNacimiento(), usuario.getSaldo());
                    sistema.getUsuarios().remove(usuario);
                    sistema.getUsuarios().add(usuarioPremium);
                    System.out.println('\n' + "El usuario " +
usuarioPremium + " ha sido promovido a premium." + '\n');
                } else {
```

```
                    System.out.println('\n' + "No se ha
encontrado un usuario con ese dni que pueda ser promovido a
premium." + '\n');
                }
            }
        }
    }

    public void listadoUsuariosPremium() {
        for (Usuario u : sistema.getUsuarios()) {
            List<Alquiler> historial = u.getHistorialViajes();
            LocalDateTime ahora = LocalDateTime.now();

            // Condición 1: 15 alquileres en el último mes
            int viajesUltimoMes = (int) historial.stream()
                    .filter(a ->
a.getFechaHoraFin().isAfter(ahora.minusMonths(1)))
                    .count();

            // Condición 2: 10 alquileres por mes durante 3
meses consecutivos
            boolean cumple3Meses = true;
            for (int i = 1; i <= 3; i++) {
                final int mesOffset = i;
                int viajesMes = (int) historial.stream()
                        .filter(a -> {
                            LocalDateTime inicioMes =
ahora.minusMonths(mesOffset);
                            LocalDateTime finMes =
ahora.minusMonths(mesOffset - 1);
                            return
a.getFechaHoraFin().isAfter(inicioMes) &&
a.getFechaHoraFin().isBefore(finMes);
                        })
                        .count();
                if (viajesMes < 10) {
                    cumple3Meses = false;
                    break;
                }
            }

            // Condición 3: Usar moto, bicicleta y patinete al
menos una vez cada uno durante 6 meses seguidos
            boolean cumple6Meses = true;
            for (int i = 6; i >= 1; i--) {
```

```
                final int mesOffset = i;
                Set<String> tiposUsados = new HashSet<>();
                historial.stream()
                        .filter(a -> {
                            LocalDateTime inicioMes =
ahora.minusMonths(mesOffset);
                            LocalDateTime finMes =
ahora.minusMonths(mesOffset - 1);
                            return
a.getFechaHoraFin().isAfter(inicioMes) &&
a.getFechaHoraFin().isBefore(finMes);
                        })
                        .forEach(a ->
tiposUsados.add(a.getVehiculo().getClass().getSimpleName()));
                if (!(tiposUsados.contains("Moto") &&
tiposUsados.contains("Bicicleta") &&
tiposUsados.contains("Patinete"))) {
                    cumple6Meses = false;
                }
            }
            // Verificamos si cumple alguna de las tres
condiciones
            if (viajesUltimoMes >= 15 || cumple3Meses ||
cumple6Meses) {
                if
(!sistema.getPosiblesUsuariosPremium().contains(u)) {
                    sistema.getPosiblesUsuariosPremium().add(u);
                }
            }
        }
    }

    public void mostrarPosiblesUsuaiosPremium() {

sistema.mostrarLista(sistema.getPosiblesUsuariosPremium(),
"posibles usuarios premium");
    }

    // Interfaces funcionales:
    interface VehiculoCreator<T> {
        T crear(Map<String, Object> datos);
    }

    interface contructorPersona {
```

```
        void construir(String dni, String nombre, int
fechaNacimiento);
    }

    interface DniExtractor<T> {
        String getDni(T obj);
    }

    interface IdExtractor<T> {
        int getId(T obj);
    }

    private <T extends Vehiculo> void
crearVehiculoGenerico(String tipoVehiculo, VehiculoCreator<T>
constructor) {
        Map<String, Object> datos = crearVehiculo(tipoVehiculo);

        if (datos != null) {
            try {
                T nuevo = constructor.crear(datos);
                boolean agregado = false;
                for (Vehiculo v : sistema.getVehiculos()) {
                    if (v.getId() == nuevo.getId()) {
                        agregado = true;
                    }
                }
                if (!agregado) {
                    sistema.getVehiculos().add(nuevo);
                    if
(tipoVehiculo.equalsIgnoreCase("Patinete") ||
tipoVehiculo.equalsIgnoreCase("Bicicleta")) {
                        agregarABase(nuevo);
                    }
                    System.out.println(tipoVehiculo + " creado: "
+ nuevo);
                } else {
                    System.out.println("Ya existe un vehiculo con
ese id");
                }
            } catch (IllegalArgumentException e) {
                System.err.println("Error al crear el/la " +
tipoVehiculo + ": " + e.getMessage());
            }
        } else {
```

```
            System.out.println("No se pudo crear el/la " +
tipoVehiculo + " por error en los datos.");
        }
    }

    private <T extends Persona> void editarPersona(List<T>
lista, String tipo) {
        sistema.mostrarLista(lista, tipo);
        System.out.println("Introduce el dni del " + tipo + " que
quieres editar");
        String dni = scanner.nextLine();
        T personaEditar = null;

        for (T p : lista) {
            if (p.getDni().equalsIgnoreCase(dni)) {
                personaEditar = p;
                break;
            }
        }

        boolean editorSaldo = false;
        if (personaEditar == null) {
            System.out.println("No se ha encontrado un " + tipo +
" con ese DNI.");
            return;
        } else {
            if (personaEditar instanceof Usuario) {
                editorSaldo = true;
            }
        }

        String opcion = "a";
        while (!opcion.trim().isEmpty()) {
            System.out.println('\n' +
"-----------------------------------------------------------------
----------------------------" + '\n');
            System.out.println(tipo.substring(0, 1).toUpperCase()
+ tipo.substring(1) + " encontrado: " + personaEditar);
            System.out.println("¿Qué desea editar?" + '\n' +
                    "1 - Nombre" + '\n' +
                    "2 - Fecha de nacimiento" + '\n' +
                    (editorSaldo ? "3 - Saldo" : "") + '\n' +
                    "Pulse Enter para salir");
            opcion = scanner.nextLine();
            if (!opcion.trim().isEmpty()) {
```

```
                try {
                    if (opcion.equalsIgnoreCase("1")) {
                        System.out.println("Introduce el nuevo
nombre:");
                        String nuevoNombre = scanner.nextLine();
                        personaEditar.setNombre(nuevoNombre);
                        System.out.println("Nombre
actualizado.");
                    } else if (opcion.equalsIgnoreCase("2")) {
                        System.out.println("Introduce la nueva
fecha de nacimiento:");
                        try {
                            int nuevaFecha =
Integer.parseInt(scanner.nextLine());

personaEditar.setfNacimiento(nuevaFecha); // Cambiar la fecha
                            System.out.println("Fecha de
nacimiento actualizada.");
                        } catch (NumberFormatException e) {
                            System.err.println("Fecha
inválida.");
                        }
                    } else if (opcion.equalsIgnoreCase("3") &&
editorSaldo) {
                        System.out.println("Introduce el nuevo
saldo:");
                        try {
                            Usuario usuarioEditar = (Usuario)
personaEditar;
                            double nuevoSaldo =
Double.parseDouble(scanner.nextLine());
                            usuarioEditar.setSaldo(nuevoSaldo);
// Cambiar el saldo
                            System.out.println("Saldo
actualizado.");
                        } catch (NumberFormatException e) {
                            System.err.println("Saldo
inválido.");
                        }
                    } else {
                        System.out.println("No ha introducido una
opción correcta");
                    }
                } catch (IllegalArgumentException e) {
```

```
                    System.out.println("Error al editar el
usuario: " + e.getMessage());
                }
            } else {
                System.out.println("Saliendo...");
            }
        }
    }

    private void crearPersona(String tipo, contructorPersona
builder) {
        System.out.println("Introduce el dni del " + tipo);
        String dni = scanner.nextLine();
        System.out.println("Introduce el nombre del " + tipo);
        String nombre = scanner.nextLine();
        try {
            System.out.println("Introduce la fecha de nacimiento
del " + tipo);
            int fechaNacimiento =
Integer.parseInt(scanner.nextLine());
            builder.construir(dni, nombre, fechaNacimiento);
        } catch (IllegalArgumentException e) {
            System.err.println("Error al crear el " + tipo + ": "
+ e.getMessage());
        }
    }

    private <T> void eliminarPorDni(List<T> lista,
DniExtractor<T> extractor, String tipo) {
        sistema.mostrarLista(lista, tipo);
        System.out.println("Introduce el dni del " + tipo + " que
quiere eliminar");
        String dni = scanner.nextLine();
        boolean eliminado = lista.removeIf(e ->
extractor.getDni(e).equalsIgnoreCase(dni));
        if (eliminado) {
            System.out.println(tipo.substring(0, 1).toUpperCase()
+ tipo.substring(1) + " eliminado");
        } else {
            System.out.println("No se ha encontrado el " + tipo +
" con DNI: " + dni);
        }
    }
```

```
    private <T> void eliminarPorId(List<T> lista, IdExtractor<T>
extractor, String tipo) {
        sistema.mostrarLista(lista, tipo);
        System.out.println("Introduce el id del " + tipo + " que
quiere eliminar");
        try {
            int id = Integer.parseInt(scanner.nextLine());
            boolean eliminado = lista.removeIf(e ->
extractor.getId(e) == id);
            if (eliminado) {
                System.out.println(tipo.substring(0,
1).toUpperCase() + tipo.substring(1) + " eliminado");
            } else {
                System.out.println("No se ha encontrado el " +
tipo);
            }
        } catch (Exception e) {
            System.out.println('\n' +"Entrada inválida. " +
'\n');
        }
    }

}
```

9. Mecánico:

```
import java.util.ArrayList;

public class Mecanico extends Empleado
{
    SistemaGestion sistema = SistemaGestion.getInstancia();
    private ArrayList<Vehiculo> vehiculosAsignados;
    private ArrayList<Base> basesAsignadas;

    public Mecanico(String dni, String nombre, int fNacimiento)
throws IllegalArgumentException
    {
        super(dni, nombre, fNacimiento);
        this.vehiculosAsignados = new ArrayList<>();
        this.basesAsignadas = new ArrayList<>();
    }

    public ArrayList<Vehiculo> getVehiculosAsignados() {
        return vehiculosAsignados;
    }
```

```
    public ArrayList<Base> getBasesAsignadas() {
        return basesAsignadas;
    }

    public void gestionarTareas(){
        sistema.gestionarTareasMecanico(this);
    }

    public void agregarVehiculoAsignado(Vehiculo vehiculo){
        vehiculosAsignados.add(vehiculo);
    }

    public void agregaBaseAsignada(Base base){
        basesAsignadas.add(base);
    }

    public void verVehiculosAsignados(){
        System.out.println("Tiene asignados los vehiculos : " +
vehiculosAsignados);
    }

    public void verBasesAsignadas(){
        System.out.println("Tiene asignados las bases: " +
basesAsignadas);
    }

    public void administrarVehiculosAsignados(){
        sistema.administrarVehiculosAsignados(this);
    }

    public void administrarBasesAsignadas(){
        sistema.administrarBasesAsignadas(this);
    }

    public Vehiculo repararVehiculo(Vehiculo vehiculo){
        vehiculo.setEstado("disponible");
        return vehiculo;
    }

    public Vehiculo repararBase(Vehiculo vehiculo){
        vehiculo.setEstado("disponible");
        return vehiculo;
    }
```

```
    public void devolverVehiculo(Vehiculo vehiculo){
        sistema.devolverVehiculoMeca(vehiculo, this);
    }

    @Override
    public String toString() {
        return "Mecanico{" +
                "dni='" + dni + '\'' +
                ", nombre='" + nombre + '\'' +
                ", fNacimiento=" + fNacimiento +
                ", vehiculosAsignados=" + vehiculosAsignados +
                ", basesAsignadas=" + basesAsignadas +
                '}';
    }
}
```

10. GestorMecanico:

```
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Scanner;

public class GestorMecanico {

    Scanner scanner = new Scanner(System.in);
    SistemaGestion sistema;

    public GestorMecanico(SistemaGestion sistema) {
        this.sistema = sistema;
    }

    public void gestionarTareasMecanico(Mecanico mecanico) {

System.out.println("\n-----------------------------------------
-----------------------------------------------\n");
        System.out.println("Bienvenido " + mecanico.getNombre());
        String accion;

        do {
            mostrarMenuMecanico();
            accion = scanner.nextLine().trim();

            if (accion.isEmpty()) {
                System.out.println("Saliendo...");
            } else {
```

```
                switch (accion) {
                    case "1":
                        mecanico.verVehiculosAsignados();
                        mecanico.verBasesAsignadas();
                        break;
                    case "2":
                        if
(mecanico.getVehiculosAsignados().isEmpty()) {
                            System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
                            System.out.println("No hay vehiculos
asignados");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
                        }else{

mecanico.administrarVehiculosAsignados();
                        }
                        break;
                    case "3":
                        if
(mecanico.getBasesAsignadas().isEmpty()) {
                            System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
                            System.out.println("No hay bases
asignadas");

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
                        }else{

mecanico.administrarBasesAsignadas();
                        }
                        break;
                    default:
                        System.out.println("No ha introducido una
opción correcta");
                }
            }
        } while (!accion.trim().isEmpty());
    }
```

```
    private void mostrarMenuMecanico() {
        System.out.println("¿Qué desea realizar?");
        System.out.println("1- Ver vehículos y bases asignados");
        System.out.println("2- Administrar vehículos asignados");
        System.out.println("3- Administrar bases asignadas");
        System.out.println("Pulse Enter para salir\n");
    }

    public void administrarVehiculosAsignados(Mecanico mecanico)
{
        try {

sistema.mostrarLista(mecanico.getVehiculosAsignados(),
"vehiculos asignados");
            Vehiculo vehiculoAdministrar = null;
            while (vehiculoAdministrar == null) {
                System.out.println("Introduzca el id del vehiculo
que desea administrar: ");
                int idVehiculo =
Integer.parseInt(scanner.nextLine());
                for (Vehiculo v :
mecanico.getVehiculosAsignados()) {
                    if (v.getId() == idVehiculo) {
                        vehiculoAdministrar = v;
                    }
                }
                if (vehiculoAdministrar == null) {
                    System.out.println("No se ha encontrado el
vehiculo con id " + idVehiculo);
                } else {
                    if
(vehiculoAdministrar.getEstado().toString().equalsIgnoreCase("a
veriado")) {
                        System.out.println("El vehiculo con id "
+ vehiculoAdministrar.getId() + " esta averiado ");
                        System.out.println("¿Qué desea hacer con
el vehiculo ? " + '\n' +
                                "1- Reparar vehiculo " + '\n' +
                                "Pulse Enter para salir" +
'\n');
                        String accion = "8";
                        while (!accion.trim().isEmpty()) {
                            accion = scanner.nextLine();
                            if (!accion.trim().isEmpty()) {
                                if (accion.equals("1")) {
```

```
                                    vehiculoAdministrar =
mecanico.repararVehiculo(vehiculoAdministrar);
                                    System.out.println("El
vehiculo con id " + vehiculoAdministrar.getId() + " ha sido
reparado");
                                    accion = "";
                                } else {
                                    System.out.println("No ha
introducido una opción correcta");
                                }
                            } else {

System.out.println("Saliendo...");
                            }
                        }
                    } else {
                        System.out.println("El vehiculo con id "
+ vehiculoAdministrar.getId() + " esta listo para su uso");
                        System.out.println("¿Qué desea hacer con
el vehiculo ? " + '\n' +
                                "1- Devolver vehiculo " + '\n' +
                                "Pulse Enter para salir" +
'\n');
                        String accion = "8";
                        while (!accion.trim().isEmpty()) {
                            accion = scanner.nextLine();
                            if (!accion.trim().isEmpty()) {
                                switch (accion) {
                                    case "1":

mecanico.devolverVehiculo(vehiculoAdministrar);
                                        break;
                                    default:
                                        System.out.println("No ha
introducido una opción correcta");
                                }
                            } else {

System.out.println("Saliendo...");
                            }
                        }
                    }

                }
            }
```

```
        } catch (NumberFormatException e) {
            System.err.println("Entrada inválida.");
        }
    }

    public void administrarBasesAsignadas(Mecanico mecanico) {
        try {
            if (mecanico.getBasesAsignadas().isEmpty()) {
                System.out.println("No hay bases asignadas");
            } else {

sistema.mostrarLista(mecanico.getBasesAsignadas(), "bases
asignadas");
                Base baseAdministrar = null;
                while (baseAdministrar == null) {
                    System.out.println("Introduzca el id de la
base que desea administrar: ");
                    int idBase =
Integer.parseInt(scanner.nextLine());
                    for (Base b : mecanico.getBasesAsignadas())
{
                        if (b.getId() == idBase) {
                            baseAdministrar = b;
                        }
                    }
                    if (baseAdministrar == null) {
                        System.out.println("No se ha encontrado
el vehiculo con id " + idBase);
                    } else {
                        if
(baseAdministrar.getTieneFallosMecanicos()) {
                            System.out.println("La base con id "
+ baseAdministrar.getId() + " esta averiada ");
                            System.out.println("¿Qué desea hacer
con la base ? " + '\n' +
                                    "1- Reparar base " + '\n' +
                                    "Pulse Enter para salir" +
'\n');
                            String accion = "8";
                            while (!accion.trim().isEmpty()) {
                                accion = scanner.nextLine();
                                if (!accion.trim().isEmpty()) {
                                    if (accion.equals("1")) {

baseAdministrar.setTieneFallosMecanicos(false);
```

```
System.out.println("Introduzca el importe de la factura: ");
                                        int importe =
Integer.parseInt(scanner.nextLine());
                                        LocalDateTime
fechaActual = LocalDateTime.now();

                                        int fechaHoraInt =
Math.toIntExact((int) fechaActual.getMonthValue() * 100000000
                                                +
fechaActual.getDayOfMonth() * 1000000
                                                +
fechaActual.getHour() * 10000
                                                +
fechaActual.getMinute() * 100);
                                        Factura factura = new
Factura(fechaHoraInt, mecanico, null, baseAdministrar, importe,
fechaActual);

sistema.getFacturas().add(factura);
                                        System.out.println("La
base con id " + baseAdministrar.getId() + " ha sido reparado");
                                        accion = "";
                                    } else {
                                        System.out.println("No ha
introducido una opción correcta");
                                    }
                                } else {

System.out.println("Saliendo...");
                                }
                            }
                        }
                    }
                }
            } catch (NumberFormatException e) {
                System.err.println("Entrada inválida.");
            }
    }

    public void devolverVehiculoMeca(Vehiculo vehiculo, Mecanico
mecanico) {
        try {
```

```
            System.out.println("Para devolver el vehiculo
introduzca donde quiere dejarlo, primero la coordenada X");
            int coordX = Integer.parseInt(scanner.nextLine());
            System.out.println("Ahora la coordenada Y");
            int coordY = Integer.parseInt(scanner.nextLine());
            if (vehiculo instanceof Moto) {
                ((Moto) vehiculo).setCoordX(coordX);
                ((Moto) vehiculo).setCoordY(coordY);

mecanico.getVehiculosAsignados().remove(vehiculo);
                System.out.println("Introduzca el importe de la
factura: ");
                int importe =
Integer.parseInt(scanner.nextLine());
                LocalDateTime fechaActual = LocalDateTime.now();

                int fechaHoraInt = Math.toIntExact((int)
fechaActual.getYear() * 10000000000L
                        + fechaActual.getMonthValue() *
100000000
                        + fechaActual.getDayOfMonth() * 1000000
                        + fechaActual.getHour() * 10000
                        + fechaActual.getMinute() * 100);
                Factura factura = new Factura(fechaHoraInt,
mecanico, vehiculo, null, importe, fechaActual);
                sistema.getFacturas().add(factura);
                System.out.println("El vehiculo con id " +
vehiculo.getId() + " se ha devuelto");
            } else {
                ArrayList<Base> basesdisp =
sistema.ordenarBases(sistema.getBases(), coordX, coordY);
                System.out.println("Este es el listado de las
bases mas cercanas a la ubicación introducida: ");
                for (Base b : basesdisp) {
                    System.out.println(b);
                }

                Base baseEscogida = null;
                while (baseEscogida == null) {
                    System.out.println('\n' + "Seleccione el id
de la base en la que va a dejar el vehiculo: " + '\n');
                    int id =
Integer.parseInt(scanner.nextLine());
                    for (Base b : basesdisp) {
                        if (b.getId() == id) {
```

```
                            baseEscogida = b;
                        }
                    }
                    if (baseEscogida != null) {

baseEscogida.agregarVehiculoDisponible(vehiculo);

mecanico.getVehiculosAsignados().remove(vehiculo);
                        System.out.println("Introduzca el importe
de la factura: ");
                        int importe =
Integer.parseInt(scanner.nextLine());
                        LocalDateTime fechaActual =
LocalDateTime.now();

                        int fechaHoraInt = Math.toIntExact((int)
fechaActual.getYear() * 10000000000L
                                + fechaActual.getMonthValue() *
100000000
                                + fechaActual.getDayOfMonth() *
1000000
                                + fechaActual.getHour() * 10000
                                + fechaActual.getMinute() *
100);
                        Factura factura = new
Factura(fechaHoraInt, mecanico, vehiculo, baseEscogida,
importe, fechaActual);
                        sistema.getFacturas().add(factura);
                        System.out.println("El vehiculo con id "
+ vehiculo.getId() + " se ha devuelto a la base " +
baseEscogida.getId());
                    } else {
                        System.out.println("No se ha encontrado
la base con id " + id);
                    }
                }
            }
        } catch (NumberFormatException e) {
            System.err.println("Entrada inválida.");
        }

    }
}
```

11. Mantenimiento:

```
import java.util.ArrayList;

public class Mantenimiento extends Empleado
{
   SistemaGestion sistema = SistemaGestion.getInstancia();
   private ArrayList<Vehiculo> vehiculosAsignados;
   private ArrayList<Vehiculo> vehiculosRecogidos;

   public Mantenimiento(String dni, String nombre, int
fNacimiento) throws IllegalArgumentException
   {
       super(dni, nombre, fNacimiento);
       this.vehiculosAsignados = new ArrayList<>();
       this.vehiculosRecogidos = new ArrayList<>();
   }

   public ArrayList<Vehiculo> getVehiculosAsignados() {
       return vehiculosAsignados;
   }

   public ArrayList<Vehiculo> getVehiculosRecogidos() {
       return vehiculosRecogidos;
   }

   public void gestionarTareas(){
       sistema.gestionarTareasMantenimiento(this);
   }

   public void agregarVehiculoAsignado(Vehiculo vehiculo){
       vehiculosAsignados.add(vehiculo);
   }

   public void agregarVehiculoRecogido(Vehiculo vehiculo){
       vehiculosRecogidos.add(vehiculo);
   }

   public void verVehiculosAsignados(){
       System.out.println("Tiene asignados: " +
vehiculosAsignados);
   }

   public void recogerVehiculo(){
       sistema.recogerVehiculo(this);
   }
```

```
    public void administrarVehiculosRecogidos(){
        sistema.administrarVehiculosRecogidos(this);
    }

    public void devolverVehiculo(Vehiculo vehiculo){
        sistema.devolverVehiculoMant(vehiculo, this);
    }

    @Override
    public String toString() {
        return "Empleado de mantenimiento{" +
                "vehiculosAsignados=" + vehiculosAsignados +
                ", dni='" + dni + '\'' +
                ", nombre='" + nombre + '\'' +
                ", fNacimiento=" + fNacimiento +
                '}';
    }
}
```

12. GestorMantenimiento:

```
import java.util.ArrayList;
import java.util.Scanner;

public class GestorMantenimiento {

    Scanner scanner = new Scanner(System.in);
    SistemaGestion sistema;

    public GestorMantenimiento(SistemaGestion sistema) {
        this.sistema = sistema;
    }

    public void gestionarTareasMantenimiento(Mantenimiento
mantenimiento) {

System.out.println("\n-----------------------------------------
-----------------------------------------\n");
        System.out.println("Bienvenido " +
mantenimiento.getNombre());
        String accion;

        do {
            mostrarMenuMantenimiento();
```

```
            accion = scanner.nextLine().trim();

            if (accion.isEmpty()) {
                System.out.println("Saliendo...");
            } else {
                switch (accion) {
                    case "1":
                        if
(mantenimiento.getVehiculosAsignados().isEmpty()) {
                            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
                            System.out.println("No hay vahiculos
asignados");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
                        }else{

mantenimiento.verVehiculosAsignados();
                        }
                        break;
                    case "2":
                        if
(mantenimiento.getVehiculosAsignados().isEmpty()) {
                            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
                            System.out.println("No hay vahiculos
asignados");

System.out.println("-----------------------------------------------
-----------------------------------------" + '\n');
                        }else{
                            mantenimiento.recogerVehiculo();
                        }
                        break;
                    case "3":
                        if
(mantenimiento.getVehiculosRecogidos().isEmpty()) {
                            System.out.println('\n' +
"-----------------------------------------------------------------
---------------------------");
                            System.out.println("No hay vahiculos
recogidos");
```

```
System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
                        }else{

mantenimiento.administrarVehiculosRecogidos();
                        }
                        break;
                    default:
                        System.out.println("No ha introducido una
opción correcta");
                }
            }

        } while (!accion.trim().isEmpty());
    }

    private void mostrarMenuMantenimiento() {
        System.out.println("¿Qué desea realizar?");
        System.out.println("1- Ver vehículos asignados");
        System.out.println("2- Recoger vehículos asignados");
        System.out.println("3- Administrar vehículos recogidos");
        System.out.println("Pulse Enter para salir\n");
    }

    public void recogerVehiculo(Mantenimiento mantenimiento) {
        if (mantenimiento.getVehiculosAsignados().isEmpty()) {
            System.out.println("No tiene vehiculos asignados");
        } else {
            mantenimiento.verVehiculosAsignados();
            System.out.println("Introduzca el id del vehiculo que
desea recoger: ");
            Vehiculo vehiculoRecoger = null;
            while (vehiculoRecoger == null) {
                int idVehiculo = -1;
                try {
                    idVehiculo =
Integer.parseInt(scanner.nextLine());
                } catch (IllegalArgumentException e) {
                    System.out.println("Entrada invalida.");
                }

                for (Vehiculo v : sistema.getVehiculos()) {
                    if (v.getId() == idVehiculo) {
                        vehiculoRecoger = v;
```

```
                    }
                }
                if (vehiculoRecoger == null) {
                    System.out.println("No se ha encontrado el
vehiculo con id " + idVehiculo);
                } else {

                    if (vehiculoRecoger instanceof Moto) {
                        ((Moto) vehiculoRecoger).setCoordX(-1);
                        ((Moto) vehiculoRecoger).setCoordY(-1);

mantenimiento.getVehiculosAsignados().remove(vehiculoRecoger);

mantenimiento.agregarVehiculoRecogido(vehiculoRecoger);
                        System.out.println("El vehiculo con id "
+ vehiculoRecoger.getId() + " se ha recogido");
                    } else {
                        Base base = null;
                        for (Base b : sistema.getBases()) {
                            if
(b.getVehiculosDisponibles().contains(vehiculoRecoger)) {
                                base = b;
                            }
                        }
                        if (base != null) {

base.eliminarVehiculoDisponible(vehiculoRecoger);

mantenimiento.getVehiculosAsignados().remove(vehiculoRecoger);

mantenimiento.agregarVehiculoRecogido(vehiculoRecoger);
                            System.out.println("El vehiculo con
id " + vehiculoRecoger.getId() + " se ha recogido en la base
con id " + base.getId());
                        } else {
//                            System.out.println("El vehiculo
con id " + vehiculoRecoger.getId() + " no se encuentra en la
base " + base);
                            System.out.println("No se ha podido
recoger el veiculo");
                        }
                    }
                }
            }
        }
```

```
    }

    public void administrarVehiculosRecogidos(Mantenimiento
mantenimiento) {

sistema.mostrarLista(mantenimiento.getVehiculosRecogidos(),
"vehiculos recogidos");
        Vehiculo vehiculoAdministrar = null;
        while (vehiculoAdministrar == null) {
            System.out.println("Introduzca el id del vehiculo que
desea administrar: ");
            int idVehiculo = -1;
            try {
                idVehiculo =
Integer.parseInt(scanner.nextLine());
            } catch (IllegalArgumentException e) {
                System.out.println("Entrada invalida.");
            }
            for (Vehiculo v :
mantenimiento.getVehiculosRecogidos()) {
                if (v.getId() == idVehiculo) {
                    vehiculoAdministrar = v;
                }
            }
            if (vehiculoAdministrar == null) {
                System.out.println("No se ha encontrado el
vehiculo con id " + idVehiculo);
            } else {
                if
(vehiculoAdministrar.getEstado().toString().equalsIgnoreCase("r
eservado")) {
                    System.out.println("El vehiculo con id " +
vehiculoAdministrar.getId() + " tiene una bateria de " +
vehiculoAdministrar.getNivelBateria() + "%");
                    System.out.println("¿Qué desea hacer con el
vehiculo ? " + '\n' +
                            "1- Recargar su bateria " + '\n' +
                            "Pulse Enter para salir" + '\n');
                    String accion = "8";
                    while (!accion.trim().isEmpty()) {
                        accion = scanner.nextLine();
                        if (!accion.trim().isEmpty()) {
                            if (accion.equals("1")) {
```

```
vehiculoAdministrar.setNivelBateria(100);

vehiculoAdministrar.setEstado("disponible");
                                System.out.println("El vehiculo
con id " + vehiculoAdministrar.getId() + " ha recargado su
bateria al " + vehiculoAdministrar.getNivelBateria() + "%");
                            } else {
                                System.out.println("No ha
introducido una opción correcta");
                            }
                        } else {
                            System.out.println("Saliendo...");
                        }
                    }
                } else if
(vehiculoAdministrar.getEstado().toString().equalsIgnoreCase("a
veriado")) {
                    System.out.println("El vehiculo con id " +
vehiculoAdministrar.getId() + " esta averiado ");
                    System.out.println("¿Qué desea hacer con el
vehiculo ? " + '\n' +
                            "1- Dejarlo en manos de un mecanico
" + '\n' +
                            "Pulse Enter para salir" + '\n');
                    String accion = "8";
                    while (!accion.trim().isEmpty()) {
                        accion = scanner.nextLine();
                        if (!accion.trim().isEmpty()) {
                            if (accion.equals("1")) {

mantenimiento.getVehiculosRecogidos().remove(vehiculoAdministra
r);
                                accion = "";
                                System.out.println("El vehiculo
con id " + vehiculoAdministrar.getId() + " se ha dejado en el
taller");
                            } else {
                                System.out.println("No ha
introducido una opción correcta");
                            }
                        } else {
                            System.out.println("Saliendo...");
                        }
                    }
```

```
                } else {
                    System.out.println("El vehiculo con id " +
vehiculoAdministrar.getId() + " esta listo para su uso");
                    System.out.println("¿Qué desea hacer con el
vehiculo ? " + '\n' +
                            "1- Devolver vehiculo " + '\n' +
                            "Pulse Enter para salir" + '\n');
                    String accion = "8";
                    while (!accion.trim().isEmpty()) {
                        accion = scanner.nextLine();
                        if (!accion.trim().isEmpty()) {
                            if (accion.equals("1")) {

mantenimiento.devolverVehiculo(vehiculoAdministrar);
                                accion = "";
                            } else {
                                System.out.println("No ha
introducido una opción correcta");
                            }
                        } else {
                            System.out.println("Saliendo...");
                        }
                    }
                }

            }
        }
    }

    public void devolverVehiculoMant(Vehiculo vehiculo,
Mantenimiento mantenimiento) {
        try {
            System.out.println("Para devolver el vehiculo
introduzca donde quiere dejarlo, primero la coordenada X");
            int coordX = Integer.parseInt(scanner.nextLine());
            System.out.println("Ahora la coordenada Y");
            int coordY = Integer.parseInt(scanner.nextLine());
            if (vehiculo instanceof Moto) {
                ((Moto) vehiculo).setCoordX(coordX);
                ((Moto) vehiculo).setCoordY(coordY);

mantenimiento.getVehiculosRecogidos().remove(vehiculo);
                System.out.println("El vehiculo con id " +
vehiculo.getId() + " se ha devuelto");
            } else {
```

```
                ArrayList<Base> basesdisp =
sistema.ordenarBases(sistema.getBases(), coordX, coordY);
                System.out.println("Este es el listado de las
bases mas cercanas a la ubicación introducida: ");
                for (Base b : basesdisp) {
                    System.out.println(b);
                }

                Base baseEscogida = null;
                while (baseEscogida == null) {
                    System.out.println('\n' + "Seleccione el id
de la base en la que va a dejar el vehiculo: " + '\n');
                    int id =
Integer.parseInt(scanner.nextLine());
                    for (Base b : basesdisp) {
                        if (b.getId() == id) {
                            baseEscogida = b;
                        }
                    }
                    if (baseEscogida != null) {

baseEscogida.agregarVehiculoDisponible(vehiculo);

mantenimiento.getVehiculosRecogidos().remove(vehiculo);
                        System.out.println("El vehiculo con id "
+ vehiculo.getId() + " se ha devuelto a la base " +
baseEscogida.getId());
                    } else {
                        System.out.println("No se ha encontrado
la base con id " + id);
                    }
                }
            }
        } catch (IllegalArgumentException e) {
            System.out.println("Entrada invalida.");
        }
    }
}
```

13. Vehiculo:

```
/**
* Clase abstracta que representa un vehículo genérico.
* Define atributos comunes y métodos esenciales relacionados con
su estado y localización.
```

```
*
* @author
*/

// Enumeración que define los posibles estados de un vehículo
enum EstadosVehiculo {
   DISPONIBLE, RESERVADO, AVERIADO
}

public abstract class Vehiculo {
   // Atributos de instancia
   public int id;                      // Identificador único
del vehículo
   public double nivelBateria;        // Nivel actual de batería
(0 a 100)
   public EstadosVehiculo estado;     // Estado actual del
vehículo

   /**
    * Constructor de la clase Vehiculo.
    * Inicializa los atributos con los valores recibidos como
parámetros.
    *
    * @param id           Identificador del vehículo
    */
   public Vehiculo(int id) {
       this.id = id;
       setNivelBateria(100); // Se asegura de que el nivel de
batería no supere el 100%
       setEstado("DISPONIBLE");
   }


   // Métodos getter y setter para acceder y modificar los
atributos de la clase
   public int getId() {
       return id;
   }

   public void setId(int id) {
       this.id = id;
   }

   public double getNivelBateria() {
       return nivelBateria;
```

```
    }

    /**
     * Establece el nivel de batería del vehículo.
     * Si el valor recibido es mayor a 100, se ajusta
automáticamente a 100.
     *
     * @param nivelBateria Valor a establecer
     */
    public void setNivelBateria(double nivelBateria) {
        if (nivelBateria > 100) {
            nivelBateria = 100;
        }
        this.nivelBateria = nivelBateria;
    }

    public void modificarNivelBateria(int nivelBateria) {
        this.nivelBateria = nivelBateria;
    }

    public EstadosVehiculo getEstado() {
        return estado;
    }

    /**
     * Establece el estado del vehículo a partir de una cadena.
     * Se convierte la cadena a mayúsculas y se valida que
coincida con uno de los valores del enum.
     *
     * @param estado Estado a establecer como cadena
     */
    public void setEstado(String estado) {
        try {
            this.estado =
EstadosVehiculo.valueOf(estado.toUpperCase());
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("El estado debe
ser 'DISPONIBLE', 'RESERVADO' o 'AVERIADO'. Valor recibido: " +
estado);
        }
    }

    /**
     * Método abstracto que debe ser implementado por las
subclases.
```

```
    * Calcula el consumo de batería del vehículo en función del
tiempo de uso.
    *
    * @param minutos Tiempo de uso en minutos
    */
   public abstract void calcularConsumoBateria(int minutos);

}
```

14. Bicicleta:

```
/**
* Clase que representa una bicicleta eléctrica.
* Extiende la clase abstracta Vehiculo e implementa el
comportamiento específico
* de cálculo de consumo de batería para bicicletas.
*
* @author
*/

public class Bicicleta extends Vehiculo {

   /**
    * Constructor de la clase Bicicleta.
    * Inicializa una bicicleta con los valores especificados.
    *
    * @param id           Identificador único de la bicicleta
    * @throws IllegalArgumentException Si el estado proporcionado
no es válido
    */
   public Bicicleta(int id) throws IllegalArgumentException {
       // Llama al constructor de la clase base Vehiculo
       super(id);
   }

   /**
    * Implementación del método abstracto calcularConsumoBateria.
    * En el caso de las bicicletas eléctricas, se estima un
consumo de 1% de batería por minuto de uso.
    *
    * @param minutos Tiempo de uso en minutos
    */
   @Override
   public void calcularConsumoBateria(int minutos) {
       setNivelBateria(this.nivelBateria- (minutos * 1));
```

```
    }

    /**
     * Devuelve una representación textual del objeto Bicicleta.
     * Incluye los valores actuales de sus atributos principales.
     *
     * @return Cadena con la información formateada del objeto
Bicicleta
     */
    @Override
    public String toString() {
        return "Bicicleta{" +
                "id=" + id +
                ", nivelBateria=" + nivelBateria +
                ", estado=" + estado +
                '}';
    }
}
```

15. Patinete:

```
/**
* Clase que representa un patinete eléctrico.
* Hereda de la clase abstracta Vehiculo y proporciona una
implementación específica
* del cálculo de consumo de batería para este tipo de vehículo.
*
* @author
*/
public class Patinete extends Vehiculo {

    /**
     * Constructor de la clase Patinete.
     * Inicializa un nuevo patinete con los parámetros
especificados.
     *
     * @param id            Identificador único del patinete
     * @throws IllegalArgumentException Si el estado proporcionado
no es válido
     */
    public Patinete(int id) throws IllegalArgumentException {
        // Llama al constructor de la clase base Vehiculo
        super(id);
```

```
    }

    /**
     * Implementación del método abstracto para calcular el
consumo de batería.
     * Para los patinetes, se asume un consumo de 0.5% de batería
por minuto de uso.
     *
     * @param minutos Tiempo de uso en minutos
     * @return Consumo estimado de batería en porcentaje
     */
    @Override
    public void calcularConsumoBateria(int minutos) {
        setNivelBateria(this.nivelBateria- (minutos * 0.5));
    }

    /**
     * Devuelve una representación en cadena del objeto Patinete.
     * Incluye todos los atributos relevantes del vehículo.
     *
     * @return Cadena con la información del patinete
     */
    @Override
    public String toString() {
        return "Patinete{" +
                "id=" + id +
                ", nivelBateria=" + nivelBateria +
                ", estado=" + estado +
                '}';
    }
}
```

16. Moto hereda de vehiculo:

```
/**
* Clase que representa una moto eléctrica.
* Hereda de la clase abstracta Vehiculo e incorpora un atributo
adicional: la cilindrada.
* El consumo de batería varía en función del tipo de cilindrada
(PEQUEÑA o GRANDE).
*
* @author
*/

// Enumeración que define los posibles tipos de cilindrada para
una moto
```

```java
enum Cilindradas {
   PEQUEÑA, GRANDE
}

public class Moto extends Vehiculo {

   // Atributo adicional específico de las motos
   public Cilindradas cilindrada;
   public int coordX;                // Coordenada X de la
ubicación de la moto
   public int coordY;                // Coordenada Y de la
ubicación de la moto

   /**
    * Constructor de la clase Moto.
    * Inicializa una moto con los valores especificados.
    *
    * @param id           Identificador único de la moto
    * @param coordX       Coordenada X de ubicación
    * @param coordY       Coordenada Y de ubicación
    * @param cilindrada   Tipo de cilindrada como cadena
("PEQUEÑA" o "GRANDE")
    * @throws IllegalArgumentException Si el estado o la
cilindrada no son válidos
    */
   public Moto(int id, int coordX, int coordY, String
cilindrada) throws IllegalArgumentException {
       // Llama al constructor de la clase base Vehiculo
       super(id);
       this.coordX = coordX;
       this.coordY = coordY;
       setCilindrada(cilindrada);

   }

   public int getCoordX() {
       return coordX;
   }

   public void setCoordX(int coordX) {
       this.coordX = coordX;
   }

   public int getCoordY() {
       return coordY;
```

```
    }

    public void setCoordY(int coordY) {
        this.coordY = coordY;
    }

    /**
     * Devuelve la cilindrada de la moto.
     *
     * @return Cilindrada de la moto (enum)
     */
    public Cilindradas getCilindrada() {
        return cilindrada;
    }

    /**
     * Establece la cilindrada de la moto a partir de una cadena.
     * La cadena debe coincidir con los valores del enum:
"PEQUEÑA" o "GRANDE".
     *
     * @param cilindrada Cadena que representa el tipo de
cilindrada
     * @throws IllegalArgumentException Si el valor no es válido
     */
    public void setCilindrada(String cilindrada) {
        try {
            this.cilindrada =
Cilindradas.valueOf(cilindrada.toUpperCase());
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("El estado debe
ser 'PEQUEÑA' o 'GRANDE'. Valor recibido: " + cilindrada);
        }
    }

    /**
     * Implementación del método abstracto que calcula el consumo
de batería.
     * El consumo depende del tipo de cilindrada:
     * - PEQUEÑA: 0.4% por minuto
     * - GRANDE: 0.25% por minuto (más eficiente)
     *
     * @param minutos Tiempo de uso en minutos
     * @return Consumo estimado de batería en porcentaje
     */
    @Override
```

```
    public void calcularConsumoBateria(int minutos) {
        if (cilindrada == Cilindradas.PEQUEÑA) {
            setNivelBateria(this.nivelBateria- (minutos * 0.4));
        } else {
            setNivelBateria(this.nivelBateria- (minutos *
0.25));
        }
    }

    /**
     * Devuelve una representación textual del objeto Moto.
     * Incluye todos los atributos principales del vehículo y su
cilindrada.
     *
     * @return Cadena con la información de la moto
     */
    @Override
    public String toString() {
        return "Moto{" +
                "id=" + id +
                ", nivelBateria=" + nivelBateria +
                ", coordX=" + coordX +
                ", coordY=" + coordY +
                ", estado=" + estado +
                ", cilindrada=" + cilindrada +
                '}';
    }
}
```

17. Base:

```
import java.util.ArrayList;

public class Base
{
    // instance variables - replace the example below with your
own
    public int id;
    public int capacidad;
    public int coordX;
    public int coordY;
    private ArrayList<Vehiculo> vehiculosDisponibles = new
ArrayList();
    public int huecosDisponibles;
    public boolean tieneFallosMecanicos;
```

```java
public Base(int id, int capacidad, int coordX, int coordY)
{
    this.id = id;
    this.coordX = coordX;
    this.coordY = coordY;
    this.capacidad = capacidad;
    this.huecosDisponibles = capacidad;
    this.tieneFallosMecanicos = false;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public int getCapacidad() {
    return capacidad;
}

public void setCapacidad(int capacidad) {
    this.capacidad = capacidad;
    int ocupados = getVehiculosDisponibles().size();
    this.huecosDisponibles = capacidad-ocupados;
}

public int getCoordX() {
    return coordX;
}

public void setCoordX(int coordX) {
    this.coordX = coordX;
}

public int getCoordY() {
    return coordY;
}

public void setCoordY(int coordY) {
    this.coordY = coordY;
}

public ArrayList<Vehiculo> getVehiculosDisponibles() {
```

```
        return vehiculosDisponibles;
    }

    public ArrayList<Vehiculo> getBicicletasDisponibles() {
        ArrayList<Vehiculo> bicicletasDisponibles = new
ArrayList();
        for (Vehiculo v : vehiculosDisponibles) {
            try {
                if (v.getClass() == Class.forName("Bicicleta") &&
v.getEstado() == EstadosVehiculo.DISPONIBLE){
                    bicicletasDisponibles.add(v);
                }
            } catch (ClassNotFoundException e) {
                throw new RuntimeException(e);
            }

        }
        return bicicletasDisponibles;
    }

    public void agregarVehiculoDisponible(Vehiculo vehiculo) {
        if (this.huecosDisponibles > 0){
            this.vehiculosDisponibles.add(vehiculo);
            this.huecosDisponibles--;
        }else{
            System.out.println("No hay huecos disponibles, nose
ha podido agregar el vehiculo");
        }
    }

    public ArrayList<Vehiculo> getPatinetesDisponibles() {
        ArrayList<Vehiculo> patinetesDisponibles = new
ArrayList();
        for (Vehiculo v : vehiculosDisponibles) {
            try {
                if (v.getClass() == Class.forName("Patinete") &&
v.getEstado() == EstadosVehiculo.DISPONIBLE){
                    patinetesDisponibles.add(v);
                }
            } catch (ClassNotFoundException e) {
                throw new RuntimeException(e);
            }

        }
        return patinetesDisponibles;
```

```
    }

    public void eliminarVehiculoDisponible(Vehiculo vehiculo) {
        this.vehiculosDisponibles.remove(vehiculo);
        this.huecosDisponibles++;
    }

    public int getHuecosDisponibles() {
        return huecosDisponibles;
    }

    public void setHuecosDisponibles(int huecosDisponibles) {
        this.huecosDisponibles = huecosDisponibles;
    }

    public boolean getTieneFallosMecanicos() {
        return tieneFallosMecanicos;
    }

    public void setTieneFallosMecanicos(boolean
tieneFallosMecanicos) {
        this.tieneFallosMecanicos = tieneFallosMecanicos;
    }

    @Override
    public String toString() {
        return "Base{" +
                "id=" + id +
                ", capacidad=" + capacidad +
                ", coordX=" + coordX +
                ", coordY=" + coordY +
                ", vehiculosDisponibles=" + vehiculosDisponibles
+
                ", huecosDisponibles=" + huecosDisponibles +
                ", tieneFallosMecanicos=" + tieneFallosMecanicos
+
                '}';
    }
}
```

18. Alquiler:

```
import java.time.Duration;
import java.time.LocalDateTime;
```

```
public class Alquiler
{
   // instance variables - replace the example below with your
own
   private int idAlquiler;
   private Vehiculo vehiculo;
   private Usuario usuario;
   private Base baseInicio;
   private Base baseFin;
   private int coordenadasInicioX;
   private int coordenadasInicioY;
   private int coordenadasFinX;
   private int coordenadasFinY;
   private LocalDateTime fechaHoraInicio;
   private LocalDateTime fechaHoraFin;
   private int tiempoViaje;
   private Tarifa tarifa;
   private double importe;

   public Alquiler(int idAlquiler, Vehiculo vehiculo, Usuario
usuario, Base baseInicio, Base
           baseFin, int coordenadasInicioX, int
coordenadasInicioY, int coordenadasFinX, int coordenadasFinY,
Tarifa tarifa, LocalDateTime fechaHoraInicio,LocalDateTime
fechaHoraFin)
   {
       this.idAlquiler = idAlquiler;
       this.vehiculo = vehiculo;
       this.usuario = usuario;
       this.baseInicio = baseInicio;
       this.baseFin = baseFin;
       if (vehiculo instanceof Moto) {
           this.coordenadasInicioX = coordenadasInicioX;
           this.coordenadasInicioY = coordenadasInicioY;
           this.coordenadasFinX = coordenadasFinX;
           this.coordenadasFinY = coordenadasFinY;
       }else {
           this.coordenadasInicioX = baseInicio.getCoordX();
           this.coordenadasInicioY = baseInicio.getCoordY();
           this.coordenadasFinX = baseFin.getCoordX();
           this.coordenadasFinY = baseFin.getCoordY();
       }
       this.fechaHoraInicio = fechaHoraInicio;
       this.fechaHoraFin = fechaHoraFin;
       this.tarifa = tarifa;
```

```
        this.importe = 0;
    }

    public Alquiler(int idAlquiler, Vehiculo vehiculo, Usuario
usuario, Base baseInicio, Base
            baseFin, int coordenadasInicioX, int
coordenadasInicioY, int coordenadasFinX, int coordenadasFinY,
Tarifa tarifa)
    {
        this.idAlquiler = idAlquiler;
        this.vehiculo = vehiculo;
        this.usuario = usuario;
        this.baseInicio = baseInicio;
        this.baseFin = baseFin;
        if (vehiculo instanceof Moto) {
            this.coordenadasInicioX = coordenadasInicioX;
            this.coordenadasInicioY = coordenadasInicioY;
            this.coordenadasFinX = coordenadasFinX;
            this.coordenadasFinY = coordenadasFinY;
        }else {
            this.coordenadasInicioX = baseInicio.getCoordX();
            this.coordenadasInicioY = baseInicio.getCoordY();
            this.coordenadasFinX = baseFin.getCoordX();
            this.coordenadasFinY = baseFin.getCoordY();
        }
        this.fechaHoraInicio = LocalDateTime.now();
        this.tarifa = tarifa;
        this.importe = 0;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public Vehiculo getVehiculo() {
        return vehiculo;
    }

    public Base getBaseInicio() {
        return baseInicio;
    }

    public Base getBaseFin() {
        return baseFin;
    }
```

```
    public int getCoordenadasFinY() {
        return coordenadasFinY;
    }

    public int getCoordenadasFinX() {
        return coordenadasFinX;
    }

    public int getTiempoViaje() {
        return tiempoViaje;
    }

    public double getImporte() {
        return importe;
    }

    public LocalDateTime getFechaHoraFin() {
        return fechaHoraFin;
    }

    public void setImporte() {
        int descuentoPremium = 0;
        importe = tarifa.getPrecioPorMinuto() * tiempoViaje;
        if (usuario instanceof UsuarioPremium) {
            descuentoPremium = tarifa.getDescuentoPremium();
            importe = importe - (importe * descuentoPremium /
100);
        }
        this.importe = importe;
    }

    public void finalizarAlquiler(){
        System.out.println(fechaHoraInicio);
        fechaHoraFin = LocalDateTime.now();
        System.out.println(fechaHoraFin);
        tiempoViaje =  (int) Duration.between(fechaHoraInicio,
fechaHoraFin).toMinutes();
        setImporte();
        System.out.println("El precio del viaje han sido " +
importe + "€");
    }

    @Override
    public String toString() {
```

```
        return "Alquiler{" +
                "idAlquiler=" + idAlquiler +
//                ", vehiculo=" + vehiculo +
//                ", usuario=" + usuario +
//                ", baseInicio=" + baseInicio +
//                ", baseFin=" + baseFin +
//                ", coordenadasInicioX=" + coordenadasInicioX
+
//                ", coordenadasInicioY=" + coordenadasInicioY
+
//                ", coordenadasFinX=" + coordenadasFinX +
//                ", coordenadasFinY=" + coordenadasFinY +
                ", fechaHoraInicio=" + fechaHoraInicio +
                ", fechaHoraFin=" + fechaHoraFin +
//                ", tiempoViaje=" + tiempoViaje +
//                ", tarifa=" + tarifa +
//                ", importe=" + importe +
                '}';
    }
}
```

19. Factura:

```
import java.time.LocalDateTime;

public class Factura
{
    private int idFactura;
    private Mecanico mecanico;
    private Vehiculo vehiculo;
    private Base base;
    private double importe;
    private LocalDateTime fecha;

    public Factura(int idFactura, Mecanico mecanico, Vehiculo
vehiculo, Base base, double
            importe, LocalDateTime fecha)
    {
        this.idFactura = idFactura;
        this.mecanico = mecanico;
        this.vehiculo = vehiculo;
        this.base = base;
        this.importe = importe;
        this.fecha = fecha;
    }
```

```
    @Override
    public String toString() {
        return "Factura{" +
                "idFactura=" + idFactura +
                ", mecanico=" + mecanico +
                ", vehiculo=" + vehiculo +
                ", base=" + base +
                ", importe=" + importe +
                ", fecha=" + fecha +
                '}';
    }
}
```

20. Tarifa:

```
enum TipoVehiculos {
    MOTO, BICICLETA, PATINETE
}

public class Tarifa
{
    private TipoVehiculos tipoVehiculo;
    private int precioPorMinuto;
    private int descuentoPremium;

    public Tarifa(TipoVehiculos tipoVehiculo, int
precioPorMinuto, int descuentoPremium)
    {
        this.tipoVehiculo = tipoVehiculo;
        this.precioPorMinuto = precioPorMinuto;
        this.descuentoPremium = descuentoPremium;
    }

    public TipoVehiculos getTipoVehiculo() {
        return tipoVehiculo;
    }

    public void setTipoVehiculo(TipoVehiculos tipoVehiculo) {
        this.tipoVehiculo = tipoVehiculo;
    }

    public int getPrecioPorMinuto() {
        return precioPorMinuto;
    }
```

```
    public void setPrecioPorMinuto(int precioPorMinuto) {
        this.precioPorMinuto = precioPorMinuto;
    }

    public int getDescuentoPremium() {
        return descuentoPremium;
    }

    public void setDescuentoPremium(int descuentoPremium) {
        this.descuentoPremium = descuentoPremium;
    }

    @Override
    public String toString() {
        return '\n' + "La tarifa de " + tipoVehiculo + " es:"+
'\n' + "Precio por minuto: " + precioPorMinuto+ "€" +
                '\n' + "Descuento premium " + descuentoPremium +
"%" + '\n';
    }
}
```

21. SistemaGestion:

```
import java.time.LocalDateTime;
import java.util.*;

/**
* Write a description of class Administrador here.
*
* @author (your name)
* @version (a version number or a date)
*/

public class SistemaGestion {

    private static SistemaGestion instancia;

    Scanner scanner = new Scanner(System.in);
    // instance variables - replace the example below with your
own
    private ArrayList<Usuario> usuarios = new ArrayList();
    private ArrayList<Vehiculo> vehiculos = new ArrayList();
    private ArrayList<Mecanico> mecanicos = new ArrayList();
```

```
    private ArrayList<Mantenimiento> mantenimientos = new
ArrayList();
    private ArrayList<Base> bases = new ArrayList();
    private ArrayList<Factura> facturas = new ArrayList();
    private ArrayList<Usuario> posiblesUsuariosPremium = new
ArrayList();
    static GestorAdministrador gestorAdministrador;
    static GestorUsuario gestorUsuario;
    static GestorMecanico gestorMecanico;
    static GestorMantenimiento gestorMantenimiento;
    private int limiteX;
    private int limiteY;

    //Inicializamos las tarifas por defecto asi el administrador
no se vera obligado a crearlas
    Tarifa tarifaMoto = new Tarifa(TipoVehiculos.MOTO, 1, 10);
    Tarifa tarifaBicicleta = new Tarifa(TipoVehiculos.BICICLETA,
1, 10);
    Tarifa tarifaPatinete = new Tarifa(TipoVehiculos.PATINETE, 1,
10);

    public ArrayList<Usuario> getUsuarios() {
        return usuarios;
    }

    public ArrayList<Vehiculo> getVehiculos() {
        return vehiculos;
    }

    public ArrayList<Mecanico> getMecanicos() {
        return mecanicos;
    }

    public ArrayList<Mantenimiento> getMantenimientos() {
        return mantenimientos;
    }

    public ArrayList<Base> getBases() {
        return bases;
    }

    public ArrayList<Factura> getFacturas() {
        return facturas;
    }
```

```java
    public ArrayList<Usuario> getPosiblesUsuariosPremium() {
        return posiblesUsuariosPremium;
    }

    //Añadir a mano
    public void anadirUsuario(Usuario usuario) {
        if (!usuarios.contains(usuario)) {
            usuarios.add(usuario);
        }
    }

    public void anadirVehiculo(Vehiculo vehiculo) {
        if (!vehiculos.contains(vehiculo)) {
            vehiculos.add(vehiculo);
        }
    }

    public void anadirMecanico(Mecanico mecanico) {
        mecanicos.add(mecanico);
    }

    public void anadirMantenimiento(Mantenimiento mantenimiento)
{
        if (!mantenimientos.contains(mantenimiento)) {
            mantenimientos.add(mantenimiento);
        }
    }

    public void anadirBase(Base base) {
        if (!bases.contains(base)) {
            Base baseID = null;
            for (Base b : bases) {
                if (b.getId() == base.getId()) {
                    baseID = b;
                }
            }
            if (baseID == null) {
                bases.add(base);
            }
        }
    }

    /**
     * Constructor for objects of class SistemaGestion
     */
```

```
    public SistemaGestion() {
        limiteX = 1000;
        limiteY = 1000;
    }

    public static SistemaGestion getInstancia() {
        if (instancia == null) {
            instancia = new SistemaGestion();
            gestorAdministrador = new
GestorAdministrador(instancia);
            gestorUsuario = new GestorUsuario(instancia);
            gestorMecanico = new GestorMecanico(instancia);
            gestorMantenimiento = new
GestorMantenimiento(instancia);
        }
        return instancia;
    }

    public int getLimiteY() {
        return limiteY;
    }

    public int getLimiteX() {
        return limiteX;
    }

    public void gestionarTareasUsuario(Usuario usuario) {
        gestorUsuario.gestionarTareasUsuario(usuario);
    }

    public void agregarSaldo(Usuario usuario) {
        gestorUsuario.agregarSaldo(usuario);
    }

    public void alquilarVehiculo(int idAlquiler, Usuario
usuario) {
        gestorUsuario.alquilarVehiculo(idAlquiler, usuario);
    }

    public void finalizarAlquiler(Alquiler alquiler) {
        gestorUsuario.finalizarAlquiler(alquiler);
    }

    public void reportarProblema() {
        gestorUsuario.reportarProblema();
```

```
    }

    public void gestionarTareasMecanico(Mecanico mecanico) {
        gestorMecanico.gestionarTareasMecanico(mecanico);
    }

    public void administrarVehiculosAsignados(Mecanico mecanico)
{
        gestorMecanico.administrarVehiculosAsignados(mecanico);
    }

    public void administrarBasesAsignadas(Mecanico mecanico) {
        gestorMecanico.administrarBasesAsignadas(mecanico);
    }

    public void devolverVehiculoMeca(Vehiculo vehiculo, Mecanico
mecanico) {
        gestorMecanico.devolverVehiculoMeca(vehiculo, mecanico);
    }

    public void gestionarTareasMantenimiento(Mantenimiento
mantenimiento) {

gestorMantenimiento.gestionarTareasMantenimiento(mantenimiento);
    }

    public void recogerVehiculo(Mantenimiento mantenimiento) {
        gestorMantenimiento.recogerVehiculo(mantenimiento);
    }

    public void administrarVehiculosRecogidos(Mantenimiento
mantenimiento) {

gestorMantenimiento.administrarVehiculosRecogidos(mantenimiento)
;
    }

    public void devolverVehiculoMant(Vehiculo vehiculo,
Mantenimiento mantenimiento) {
        gestorMantenimiento.devolverVehiculoMant(vehiculo,
mantenimiento);
    }

    public void gestionarTareasAdministrador(Administrador
administrador) {
```

```
gestorAdministrador.gestionarTareasAdministrador(administrador);
    }

    public void gestionarUsuarios() {
        gestorAdministrador.gestionarUsuarios();
    }

    public void gestionarVehiculos() {
        gestorAdministrador.gestionarVehiculos();
    }

    public void gestionarTarifas() {
        gestorAdministrador.gestionarTarifas();
    }

    public void gestionarMecanicos() {
        gestorAdministrador.gestionarMecanicos();
    }

    public void asignarMecanico() {
        gestorAdministrador.asignarMecanico();
    }

    public void gestionarMantenimientos() {
        gestorAdministrador.gestionarMantenimientos();
    }

    public void asignarMantenimiento() {
        gestorAdministrador.asignarMantenimiento();
    }

    public void gestionarBases() {
        gestorAdministrador.gestionarBases();
    }

    public void promocionarUsuarioPremium() {
        gestorAdministrador.promocionarUsuarioPremium();
    }

    public ArrayList<Base> getBasesDisponibles() {
        ArrayList<Base> basesDisponibles = new ArrayList();
        for (Base b : bases) {
            if (!b.getTieneFallosMecanicos()) {
                basesDisponibles.add(b);
```

```
            }
        }
        return basesDisponibles;
    }

    public ArrayList<Base> getBasesDisponiblesConHueco() {
        ArrayList<Base> getBasesDisponiblesConHueco = new
ArrayList();
        for (Base b : getBasesDisponibles()) {
            if (b.getHuecosDisponibles() > 0) {
                getBasesDisponiblesConHueco.add(b);
            }
        }
        return getBasesDisponiblesConHueco;
    }

    public ArrayList<Base> ordenarBases(ArrayList<Base> bases,
int coordX, int coordY) {
        for (int i = 0; i < bases.size() - 1; i++) {
            for (int j = 0; j < bases.size() - 1 - i; j++) {
                Base b1 = bases.get(j);
                Base b2 = bases.get(j + 1);

                double dist1 = Math.pow(b1.getCoordX() - coordX,
2) + Math.pow(b1.getCoordY() - coordY, 2);
                double dist2 = Math.pow(b2.getCoordX() - coordX,
2) + Math.pow(b2.getCoordY() - coordY, 2);

                if (dist1 > dist2) {
                    // Intercambiar las bases
                    bases.set(j, b2);
                    bases.set(j + 1, b1);
                }
            }
        }
        return bases;
    }

    public ArrayList<Vehiculo> ordenarMotos(ArrayList<Vehiculo>
motos, int coordX, int coordY) {
        for (int i = 0; i < motos.size() - 1; i++) {
            for (int j = 0; j < motos.size() - 1 - i; j++) {
                Moto m1 = (Moto) motos.get(j);
                Moto m2 = (Moto) motos.get(j + 1);
```

```
                double dist1 = Math.pow(m1.getCoordX() - coordX,
2) + Math.pow(m1.getCoordY() - coordY, 2);
                double dist2 = Math.pow(m2.getCoordX() - coordX,
2) + Math.pow(m2.getCoordY() - coordY, 2);

                if (dist1 > dist2) {
                    // Intercambiar las bases
                    motos.set(j, m2);
                    motos.set(j + 1, m1);
                }
            }
        }
        return motos;
    }

    public void mostrarLista(List<?> lista, String tipo) {
        System.out.println('\n' +
"-------------------------------------------------------------
---------------------------");
        System.out.println("Esta es la lista de " + tipo + '\n');
        if (lista.isEmpty()) {
            System.out.println("No hay " + tipo);
        } else {
            for (Object elemento : lista) {
                System.out.println(elemento);
            }
        }

System.out.println("-----------------------------------------
-----------------------------------------" + '\n');
    }

}
```

22. Movilidad:

```
import java.time.LocalDateTime;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.ThreadLocalRandom;

/**
* Write a description of class Movilidad here.
*
* @author (your name)
```

```
* @version (a version number or a date)
*/
public class Movilidad {
   // instance variables - replace the example below with your
own
   SistemaGestion sistema = SistemaGestion.getInstancia();
   Scanner scanner = new Scanner(System.in);
   Administrador admin;

   /**
    * Constructor for objects of class Movilidad
    */
   public Movilidad() {
       admin = new Administrador("78153714J", "ALejando",
10101233);
       inicializarEjemplos();

   }

   public void iniciar() {
       System.out.println('\n' +
"----------------------------------------------------------------
----------------------------" + '\n');
       System.out.println("Bienvenido a la aplicacion de
movilidad");
       String accion = "1";
       while (!accion.trim().isEmpty()) {

           System.out.println("Indique el rol que tiene en esta
aplicación" + '\n' +
                   "1- Administrador " + '\n' +
                   "2- Usuario " + '\n' +
                   "3- Mecanico " + '\n' +
                   "4- Empleado de mantenimiento " + '\n' +
                   "Pulse Enter para salir" + '\n');

           accion = scanner.nextLine();
           if (!accion.trim().isEmpty()) {
               switch (accion) {
                   case "1":
                       admin.gestionarTareas();
                       break;
                   case "2":

sistema.mostrarLista(sistema.getUsuarios(), "usuarios");
```

```
                        Usuario usuarioSeleccionado = null;
                        while (usuarioSeleccionado == null) {
                            System.out.println("Introduzca su
dni: ");
                            String dni = scanner.nextLine();
                            for (Usuario u :
sistema.getUsuarios()) {
                                if
(u.getDni().equalsIgnoreCase(dni)) {
                                    usuarioSeleccionado = u;
                                }
                            }
                            if (usuarioSeleccionado == null) {
                                System.out.println("No se ha
encontrado un usuario con dni: " + dni);
                            } else {

usuarioSeleccionado.gestionarTareas();
                            }
                        }
                        break;
                    case "3":

sistema.mostrarLista(sistema.getMecanicos(), "mecanicos");
                        Mecanico mecanicoSeleccionado = null;
                        while (mecanicoSeleccionado == null) {
                            System.out.println("Introduzca su
dni: ");
                            String dni = scanner.nextLine();
                            for (Mecanico m :
sistema.getMecanicos()) {
                                if
(m.getDni().equalsIgnoreCase(dni)) {
                                    mecanicoSeleccionado = m;
                                }
                            }
                            if (mecanicoSeleccionado == null) {
                                System.out.println("No se ha
encontrado un mecanico con dni: " + dni);
                            } else {

mecanicoSeleccionado.gestionarTareas();
                            }
                        }
                        break;
```

```
                    case "4":

sistema.mostrarLista(sistema.getMantenimientos(),
"mantenimientos");
                        Mantenimiento mantenimientoSeleccionado
= null;
                        while (mantenimientoSeleccionado ==
null) {
                            System.out.println("Introduzca su
dni: ");
                            String dni = scanner.nextLine();
                            for (Mantenimiento m :
sistema.getMantenimientos()) {
                                if
(m.getDni().equalsIgnoreCase(dni)) {
                                    mantenimientoSeleccionado =
m;
                                }
                            }
                            if (mantenimientoSeleccionado ==
null) {
                                System.out.println("No se ha
encontrado un empleado de mantenimiento con dni: " + dni);
                            } else {

mantenimientoSeleccionado.gestionarTareas();
                            }
                        }
                        break;
                    default:
                        System.out.println("No ha introducido una
opción correcta");
                }
            } else {
                System.out.println("Saliendo...");
            }
        }
    }

    public void inicializarEjemplos() {
        //Tiempo
        LocalDateTime ahora = LocalDateTime.now();
        LocalDateTime hace1Meses = ahora.minusMonths(1);
        LocalDateTime hace2Meses = ahora.minusMonths(2);
        LocalDateTime hace3Meses = ahora.minusMonths(3);
```

```
        long segundosDesde1Mes =
hace1Meses.toEpochSecond(java.time.ZoneOffset.UTC);
        long segundosDesde2Mes =
hace2Meses.toEpochSecond(java.time.ZoneOffset.UTC);
        long segundosDesde3Mes =
hace3Meses.toEpochSecond(java.time.ZoneOffset.UTC);
        long segundosHasta =
ahora.toEpochSecond(java.time.ZoneOffset.UTC);

        //Usuario estandar saldo positivo no puede ser premium
saldo positivo
        crearUsuarioEstandar("85080411F", "UsuarioNopremium",
10101233, 100);
        //Usuario estandar saldo negativo no puede ser premium
        crearUsuarioEstandar("37063847Z", "UsuarioNegativo",
10101233, -100);
        //Usuario estandar puede ser premium ha usado 15
vehiculos en el ultimo mes
        Usuario UsuarioPremium1 =
crearUsuarioEstandar("34111100F", "UsuarioPremium1", 10101233,
100);
        for (int i = 1; i <= 15; i++) {
            Random rand = new Random();

            int dato1 = rand.nextInt(10);
            int dato2 = rand.nextInt(100);
            int dato3 = rand.nextInt(100);


            Base b = new Base(i, dato1, dato2, dato3);

            long randomSegundos =
ThreadLocalRandom.current().nextLong(segundosDesde1Mes,
segundosHasta);
            LocalDateTime fechafin =
LocalDateTime.ofEpochSecond(randomSegundos, 0,
java.time.ZoneOffset.UTC);
            Bicicleta bici = new Bicicleta(i - 100);
            Alquiler alquiler = new Alquiler(i, bici,
UsuarioPremium1, new Base(1, 1, 1, 1), new Base(1, 1, 1, 1), 0,
0, 0, 0, new Tarifa(null, 1, 1), fechafin, fechafin);
            UsuarioPremium1.agregarViaje(alquiler);
            UsuarioPremium1.finalizarAlquiler(alquiler);
```

```
            UsuarioPremium1.setSaldo(100);
        }

        //Usuario estandar puede ser premium ha usado 3
vehiculos en los ultimos 3 meses
        Usuario UsuarioPremium2 =
crearUsuarioEstandar("59145604T", "UsuarioPremium2", 10101233,
10000);
        for (int i = 1; i <= 10; i++) {
            Random rand = new Random();

            int dato1 = rand.nextInt(10);
            int dato2 = rand.nextInt(100);
            int dato3 = rand.nextInt(100);

            Base b = new Base(i, dato1, dato2, dato3);
            // 50% de probabilidad de añadir una bicicleta

            long randomSegundos =
ThreadLocalRandom.current().nextLong(segundosDesde1Mes,
segundosHasta);
            LocalDateTime fechafin =
LocalDateTime.ofEpochSecond(randomSegundos, 0,
java.time.ZoneOffset.UTC);
            Bicicleta bici = new Bicicleta(i - 1000);
            Alquiler alquiler = new Alquiler(i, bici,
UsuarioPremium2, new Base(1, 1, 1, 1), new Base(1, 1, 1, 1), 0,
0, 0, 0, new Tarifa(null, 1, 1), fechafin, fechafin);
            UsuarioPremium2.agregarViaje(alquiler);
            UsuarioPremium2.finalizarAlquiler(alquiler);

            long randomSegundos2 =
ThreadLocalRandom.current().nextLong(segundosDesde2Mes,
segundosDesde1Mes);
            LocalDateTime fechafin2 =
LocalDateTime.ofEpochSecond(randomSegundos2, 0,
java.time.ZoneOffset.UTC);
            Bicicleta bici2 = new Bicicleta(i - 2000);
            Alquiler alquiler2 = new Alquiler(i, bici2,
UsuarioPremium2, new Base(1, 1, 1, 1), new Base(1, 1, 1, 1), 0,
0, 0, 0, new Tarifa(null, 1, 1), fechafin2, fechafin2);
            UsuarioPremium2.agregarViaje(alquiler2);
            UsuarioPremium2.finalizarAlquiler(alquiler2);
```

```
            long randomSegundos3 =
ThreadLocalRandom.current().nextLong(segundosDesde3Mes,
segundosDesde2Mes);
            LocalDateTime fechafin3 =
LocalDateTime.ofEpochSecond(randomSegundos3, 0,
java.time.ZoneOffset.UTC);
            Bicicleta bici3 = new Bicicleta(i - 3000);
            Alquiler alquiler3 = new Alquiler(i, bici3,
UsuarioPremium2, new Base(1, 1, 1, 1), new Base(1, 1, 1, 1), 0,
0, 0, 0, new Tarifa(null, 1, 1), fechafin3, fechafin3);
            UsuarioPremium2.agregarViaje(alquiler3);
            UsuarioPremium2.finalizarAlquiler(alquiler3);

            UsuarioPremium2.setSaldo(100);
        }

        //Usuario estandar puede ser premium ha usado los 3
tipos de vehiculos en los ultimos 6 meses
        Usuario UsuarioPremium3 =
crearUsuarioEstandar("99991220P", "UsuarioPremium3", 10101233,
10000);
        for (int i = 1; i <= 6; i++) {
            Random rand = new Random();

            int dato1 = rand.nextInt(10);
            int dato2 = rand.nextInt(100);
            int dato3 = rand.nextInt(100);
            int dato4 = rand.nextInt(100);
            int dato5 = rand.nextInt(100);

            Base b = new Base(i, dato1, dato2, dato3);
            // 50% de probabilidad de añadir una bicicleta

            LocalDateTime ahora1 = LocalDateTime.now();
            LocalDateTime hasta = ahora1.minusMonths(i - 1);
            LocalDateTime desde = ahora1.minusMonths(i);

            long segundosDesde =
desde.toEpochSecond(java.time.ZoneOffset.UTC);
            long segundosHasta1 =
hasta.toEpochSecond(java.time.ZoneOffset.UTC);

            long randomSegundos =
ThreadLocalRandom.current().nextLong(segundosDesde,
segundosHasta1);
```

```
            LocalDateTime fechafin =
LocalDateTime.ofEpochSecond(randomSegundos, 0,
java.time.ZoneOffset.UTC);
            Bicicleta bici = new Bicicleta(i); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
            UsuarioPremium3.agregarViaje(new Alquiler(i, bici,
UsuarioPremium3, new Base(1, 1, 1, 1), new Base(1, 1, 1, 1), 0,
0, 0, 0, new Tarifa(null, 1, 1), fechafin, fechafin));

            Patinete patin = new Patinete(i); // Usa el mismo ID
que la base, o cámbialo si quieres que sea único
            UsuarioPremium3.agregarViaje(new Alquiler(i, patin,
UsuarioPremium3, new Base(1, 1, 1, 1), new Base(1, 1, 1, 1), 0,
0, 0, 0, new Tarifa(null, 1, 1), fechafin, fechafin));

            Moto moto = new Moto(i, dato4, dato5, "grande");//
Usa el mismo ID que la base, o cámbialo si quieres que sea
único
            UsuarioPremium3.agregarViaje(new Alquiler(i, moto,
UsuarioPremium3, new Base(1, 1, 1, 1), new Base(1, 1, 1, 1), 0,
0, 0, 0, new Tarifa(null, 1, 1), fechafin, fechafin));


            UsuarioPremium3.setSaldo(100);
        }

        //Usuario premium saldo positivo
        crearUsuarioPremium("37185714G", "UsuarioPremium",
10101233, 10000);

        //Mecanico
        Mecanico mecanico1 = new Mecanico("43150989E",
"Mecanico1", 10101233);
        sistema.anadirMecanico(mecanico1);
        Mecanico mecanico2 = new Mecanico("55598868W",
"Mecanico2", 10101233);
        sistema.anadirMecanico(mecanico2);
        Mecanico mecanico3 = new Mecanico("01840881F",
"Mecanico3", 10101233);
        sistema.anadirMecanico(mecanico3);

        //Mantenimiento
        Mantenimiento mantenimiento1 = new
Mantenimiento("69108369Q", "mantenimiento1", 10101233);
        sistema.anadirMantenimiento(mantenimiento1);
```

```
        Mantenimiento mantenimiento2 = new
Mantenimiento("17656376N", "mantenimiento2", 10101233);
        sistema.anadirMantenimiento(mantenimiento2);
        Mantenimiento mantenimiento3 = new
Mantenimiento("71335156N", "mantenimiento3", 10101233);
        sistema.anadirMantenimiento(mantenimiento3);

        //Motos
        //Moto grande normal
        Moto moto1 = new Moto(11, 10, 40, "grande"); // Usa el
mismo ID que la base, o cámbialo si quieres que sea único
        sistema.anadirVehiculo(moto1);
        //Moto pequeña normal
        Moto moto2 = new Moto(12, 150, 200, "pequeña"); // Usa
el mismo ID que la base, o cámbialo si quieres que sea único
        sistema.anadirVehiculo(moto2);
        //Moto pequena sin bateria
        Moto moto3 = new Moto(13, 400, 500, "pequeña"); // Usa
el mismo ID que la base, o cámbialo si quieres que sea único
        moto3.setNivelBateria(5);
        sistema.anadirVehiculo(moto3);
        //Moto grande sin bateria
        Moto moto4 = new Moto(14, 700, 100, "pequeña"); // Usa
el mismo ID que la base, o cámbialo si quieres que sea único
        moto4.setNivelBateria(5);
        sistema.anadirVehiculo(moto4);
        //Moto grande, bateria < 20%
        Moto moto5 = new Moto(15, 900, 350, "pequeña"); // Usa
el mismo ID que la base, o cámbialo si quieres que sea único
        moto5.setNivelBateria(17);
        sistema.anadirVehiculo(moto5);
        //Moto pequeña, bateria < 20%
        Moto moto6 = new Moto(16, 600, 700, "pequeña"); // Usa
el mismo ID que la base, o cámbialo si quieres que sea único
        moto6.setNivelBateria(15);
        sistema.anadirVehiculo(moto6);
        //Moto grande averiada
        Moto moto7 = new Moto(17, 430, 740, "pequeña"); // Usa
el mismo ID que la base, o cámbialo si quieres que sea único
        moto7.setEstado("averiado");
        sistema.anadirVehiculo(moto7);
        //Moto pequeña averiada
        Moto moto8 = new Moto(18, 20, 300, "pequeña"); // Usa el
mismo ID que la base, o cámbialo si quieres que sea único
        moto8.setEstado("averiado");
```

```
        sistema.anadirVehiculo(moto8);

        //Patinetes
        //Patinete normal
        Patinete patinete1 = new Patinete(20); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
        sistema.anadirVehiculo(patinete1);
        //Patinete averiado
        Patinete patinete2 = new Patinete(21); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
        patinete2.setEstado("averiado");
        sistema.anadirVehiculo(patinete2);
        //Patinete sin bateria
        Patinete patinete3 = new Patinete(22); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
        patinete3.setNivelBateria(5);
        sistema.anadirVehiculo(patinete3);
        //Patinete bateria < 20%
        Patinete patinete4 = new Patinete(23); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
        patinete4.setNivelBateria(17);
        sistema.anadirVehiculo(patinete4);
        //Patinete normal
        Patinete patinete5 = new Patinete(24); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
        sistema.anadirVehiculo(patinete5);
        //Patinete normal
        Patinete patinete6 = new Patinete(25); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
        sistema.anadirVehiculo(patinete6);
        //Patinete normal
        Patinete patinete7 = new Patinete(26); // Usa el mismo
ID que la base, o cámbialo si quieres que sea único
        sistema.anadirVehiculo(patinete7);

        //Bicicletas
        //Bicicleta normal
        Bicicleta bici1 = new Bicicleta(30);
        sistema.anadirVehiculo(bici1);
        //Bicicleta averiada
        Bicicleta bici2 = new Bicicleta(31);
        bici2.setEstado("averiado");
        sistema.anadirVehiculo(bici2);
        //Bicicleta sin bateria
        Bicicleta bici3 = new Bicicleta(32);
```

```
        bici3.setNivelBateria(5);
        sistema.anadirVehiculo(bici3);
        //Bicicleta bateria < 20%
        Bicicleta bici4 = new Bicicleta(34); // Usa el mismo ID
que la base, o cámbialo si quieres que sea único
        bici4.setNivelBateria(12);
        sistema.anadirVehiculo(bici4);
        //Bicicleta normal
        Bicicleta bici5 = new Bicicleta(35);
        sistema.anadirVehiculo(bici5);
        //Bicicleta normal
        Bicicleta bici6 = new Bicicleta(36);
        sistema.anadirVehiculo(bici6);
        //Bicicleta normal
        Bicicleta bici7 = new Bicicleta(37);
        sistema.anadirVehiculo(bici7);


        //Bases
        //Base averiada sin vehiculos
        Base base1 = new Base(1, 45, 300, 500);
        base1.setTieneFallosMecanicos(true);
        sistema.anadirBase(base1);
        //Base averiada con vehiculos
        Base base2 = new Base(2, 4, 700, 200);
        base2.setTieneFallosMecanicos(true);
        base2.agregarVehiculoDisponible(patinete1);
        base2.agregarVehiculoDisponible(bici1);
        sistema.anadirBase(base2);
        //Base normal sin vehiculos
        Base base3 = new Base(3, 10, 758, 934);
        sistema.anadirBase(base3);
        //Base normal con vehiculos
        Base base4 = new Base(4, 7, 293, 459);
        base4.agregarVehiculoDisponible(patinete3);
        base4.agregarVehiculoDisponible(bici2);
        base4.agregarVehiculoDisponible(patinete2);
        base4.agregarVehiculoDisponible(bici7);
        sistema.anadirBase(base4);
        //Base normal con vehiculos
        Base base5 = new Base(5, 9, 563, 245);
        base5.agregarVehiculoDisponible(patinete5);
        base5.agregarVehiculoDisponible(bici4);
        base4.agregarVehiculoDisponible(patinete6);
        base4.agregarVehiculoDisponible(bici5);
```

```
        base4.agregarVehiculoDisponible(patinete7);
        sistema.anadirBase(base5);
        //Base normal con vehiculos llena
        Base base6 = new Base(6, 3, 790, 982);
        base6.agregarVehiculoDisponible(patinete4);
        base6.agregarVehiculoDisponible(bici6);
        base6.agregarVehiculoDisponible(bici3);
        sistema.anadirBase(base6);
    }

    public Usuario crearUsuarioEstandar(String dni, String
nombre, int fNacimiento, double saldo) {
        UsuarioEstandar usuario = new UsuarioEstandar(dni,
nombre, fNacimiento, saldo);
        sistema.anadirUsuario(usuario);
        return usuario;
    }

    public Usuario crearUsuarioPremium(String dni, String
nombre, int fNacimiento, double saldo) {
        UsuarioPremium usuario = new UsuarioPremium(dni, nombre,
fNacimiento, saldo);
        sistema.anadirUsuario(usuario);
        return usuario;
    }

    public void mostrarEjemplos() {
        for (int j = 1; j <= 20; j++) {
            String nombre = "Usuario" + j;
            String dni = "a" + j;
            UsuarioEstandar usuarioAleatorio = new
UsuarioEstandar(dni, nombre, 11112000, 100);
            for (int i = 1; i <= 20; i++) {
                Random rand = new Random();

                int dato1 = rand.nextInt(10); // Rango de 0 a 99
(ajusta si necesitas otro rango)
                int dato2 = rand.nextInt(100);
                int dato3 = rand.nextInt(100);
                int dato4 = rand.nextInt(100);
                int dato5 = rand.nextInt(100);

                Base b = new Base(i, dato1, dato2, dato3);
                // 50% de probabilidad de añadir una bicicleta
```

```
                if (rand.nextBoolean()) {
                    LocalDateTime ahora = LocalDateTime.now();
                    LocalDateTime hace6Meses =
ahora.minusMonths(1);

                    long segundosDesde =
hace6Meses.toEpochSecond(java.time.ZoneOffset.UTC);
                    long segundosHasta =
ahora.toEpochSecond(java.time.ZoneOffset.UTC);

                    long randomSegundos =
ThreadLocalRandom.current().nextLong(segundosDesde,
segundosHasta);
                    LocalDateTime fechafin =
LocalDateTime.ofEpochSecond(randomSegundos, 0,
java.time.ZoneOffset.UTC);
                    Bicicleta bici = new Bicicleta(i); // Usa el
mismo ID que la base, o cámbialo si quieres que sea único
                    usuarioAleatorio.agregarViaje(new
Alquiler(i, bici, usuarioAleatorio, new Base(1, 1, 1, 1), new
Base(1, 1, 1, 1), 0, 0, 0, 0, new Tarifa(null, 1, 1), fechafin,
fechafin));

                }
                if (rand.nextBoolean()) {
                    LocalDateTime ahora = LocalDateTime.now();
                    LocalDateTime hace6Meses =
ahora.minusMonths(3);

                    long segundosDesde =
hace6Meses.toEpochSecond(java.time.ZoneOffset.UTC);
                    long segundosHasta =
ahora.toEpochSecond(java.time.ZoneOffset.UTC);

                    long randomSegundos =
ThreadLocalRandom.current().nextLong(segundosDesde,
segundosHasta);
                    LocalDateTime fechafin =
LocalDateTime.ofEpochSecond(randomSegundos, 0,
java.time.ZoneOffset.UTC);
                    Patinete patin = new Patinete(i); // Usa el
mismo ID que la base, o cámbialo si quieres que sea único
                    usuarioAleatorio.agregarViaje(new
Alquiler(i, patin, usuarioAleatorio, new Base(1, 1, 1, 1), new
```

```
Base(1, 1, 1, 1), 0, 0, 0, 0, new Tarifa(null, 1, 1), fechafin,
fechafin));

                }
                if (rand.nextBoolean()) {
                    LocalDateTime ahora = LocalDateTime.now();
                    LocalDateTime hace6Meses =
ahora.minusMonths(6);

                    long segundosDesde =
hace6Meses.toEpochSecond(java.time.ZoneOffset.UTC);
                    long segundosHasta =
ahora.toEpochSecond(java.time.ZoneOffset.UTC);

                    long randomSegundos =
ThreadLocalRandom.current().nextLong(segundosDesde,
segundosHasta);
                    LocalDateTime fechafin =
LocalDateTime.ofEpochSecond(randomSegundos, 0,
java.time.ZoneOffset.UTC);
                    Moto moto = new Moto(i, dato4, dato5,
"grande");// Usa el mismo ID que la base, o cámbialo si quieres
que sea único
                    usuarioAleatorio.agregarViaje(new
Alquiler(i, moto, usuarioAleatorio, new Base(1, 1, 1, 1), new
Base(1, 1, 1, 1), 0, 0, 0, 0, new Tarifa(null, 1, 1), fechafin,
fechafin));

                }
                sistema.anadirUsuario(usuarioAleatorio);
            }
//                if (rand.nextBoolean()) {
//                    Patinete patin = new Patinete(i); // Usa
el mismo ID que la base, o cámbialo si quieres que sea único
//                    sistema.anadirVehiculo(patin);
//                    b.agregarVehiculoDisponible(patin);
//                }
//                if (rand.nextBoolean()) {
//                    Moto moto = new Moto(i, dato4, dato5,
"grande"); // Usa el mismo ID que la base, o cámbialo si
quieres que sea único
//                    sistema.anadirVehiculo(moto);
//                }
        }
    }
```

```
}
```