1. Estructura de paquetes (Controller, Service, Repository, Model): Esta estructura sigue el patrón de arquitectura en capas, que es una práctica común en el desarrollo de aplicaciones empresariales. Separa las responsabilidades y mejora la mantenibilidad y escalabilidad del código.
    - Controller (Presentación): Maneja las solicitudes HTTP y actúa como punto de entrada de la aplicación.
    - Service (Negocio): Contiene la lógica de negocio y coordina las operaciones entre el controlador y el repositorio.
    - Repository (Datos): Se encarga de la persistencia y recuperación de datos.
    - Model (Dominio): Define los objetos de dominio que representan las entidades de la aplicación.

2. Uso de Lombok (Car.java): Lombok reduce el código repetitivo (boilerplate) generando automáticamente getters, setters, constructores, etc. Esto mejora la legibilidad del código y reduce la posibilidad de errores.

   La anotación @Data de Lombok genera automáticamente todos los métodos necesarios, lo que nos ahorra tiempo y hace que el código sea más limpio.

3. Instancia del objeto en la capa de datos y logging (CarRepository.java): Esto simula la creación y recuperación de datos, y demuestra cómo se puede usar el logging para depuración y monitoreo.

   El uso de @Slf4j de Lombok simplifica la configuración del logging, y el log.info() muestra cómo se pueden registrar datos importantes para el seguimiento de la aplicación.

4. Acceso a propiedades en la capa de negocio (CarService.java): Demuestra cómo se pueden externalizar configuraciones y acceder a ellas desde la capa de servicio, lo que permite una mayor flexibilidad y facilita los cambios sin necesidad de modificar el código.

   El uso de @Value permite inyectar valores de configuración directamente desde el archivo application.properties, facilitando la gestión de configuraciones.

5. Endpoint GET y logging en la capa de presentación (CarController.java): Crea un punto de entrada RESTful para la aplicación y demuestra cómo se puede usar el logging para registrar accesos a la API.

   El uso de @GetMapping crea un endpoint RESTful, y el log.info() registra cada acceso a este endpoint, lo que es útil para monitoreo y análisis de uso de la API.