Imagen en blanco y negro

Descripción generada automáticamente con confianza baja

Laboratorio 4

Middleware- gestión de planos.

url repositorio:

[https://github.com/ARSW2023-2/ARSW\_Lab03.git](https://github.com/ARSW2023-2/ARSW_Lab04.git)

Integrantes:

Luisa Fernanda Bermúdez Girón

Karol Daniela Ladino Ladino

Squad:

Inside Out

Profesor:

Javier Iván Toquica Barrera

Curso:

ARSW – 1

Fecha De Entrega:

15-09-2023

1. **Introducción**

El laboratorio propuesto se centra en la aplicación de principios fundamentales de desarrollo de software, específicamente el Principio de Inversión de Dependencias, el uso de Contenedores Livianos como Spring y la práctica de la Inyección de Dependencias. Para ello, se divide en dos partes.

En la primera parte, se aborda la configuración de una aplicación de análisis de textos que requiere un verificador gramatical con un corrector ortográfico. A través del uso de Spring y anotaciones, se ilustra cómo gestionar dependencias y seleccionar implementaciones específicas en tiempo de ejecución.

La segunda parte se enfoca en la construcción de un modelo de clases para una aplicación de gestión de planos arquitectónicos. Aquí, se profundiza en la implementación de un esquema de inyección de dependencias y se introduce la capacidad de aplicar filtros a los planos antes de su consulta, demostrando cómo la Inyección de Dependencias puede mejorar la adaptabilidad de una aplicación a diferentes requisitos.

1. **Desarrollo del laboratorio**

**Taller – Principio de Inversión de dependencias, Contenedores Livianos e Inyección de dependencias.**

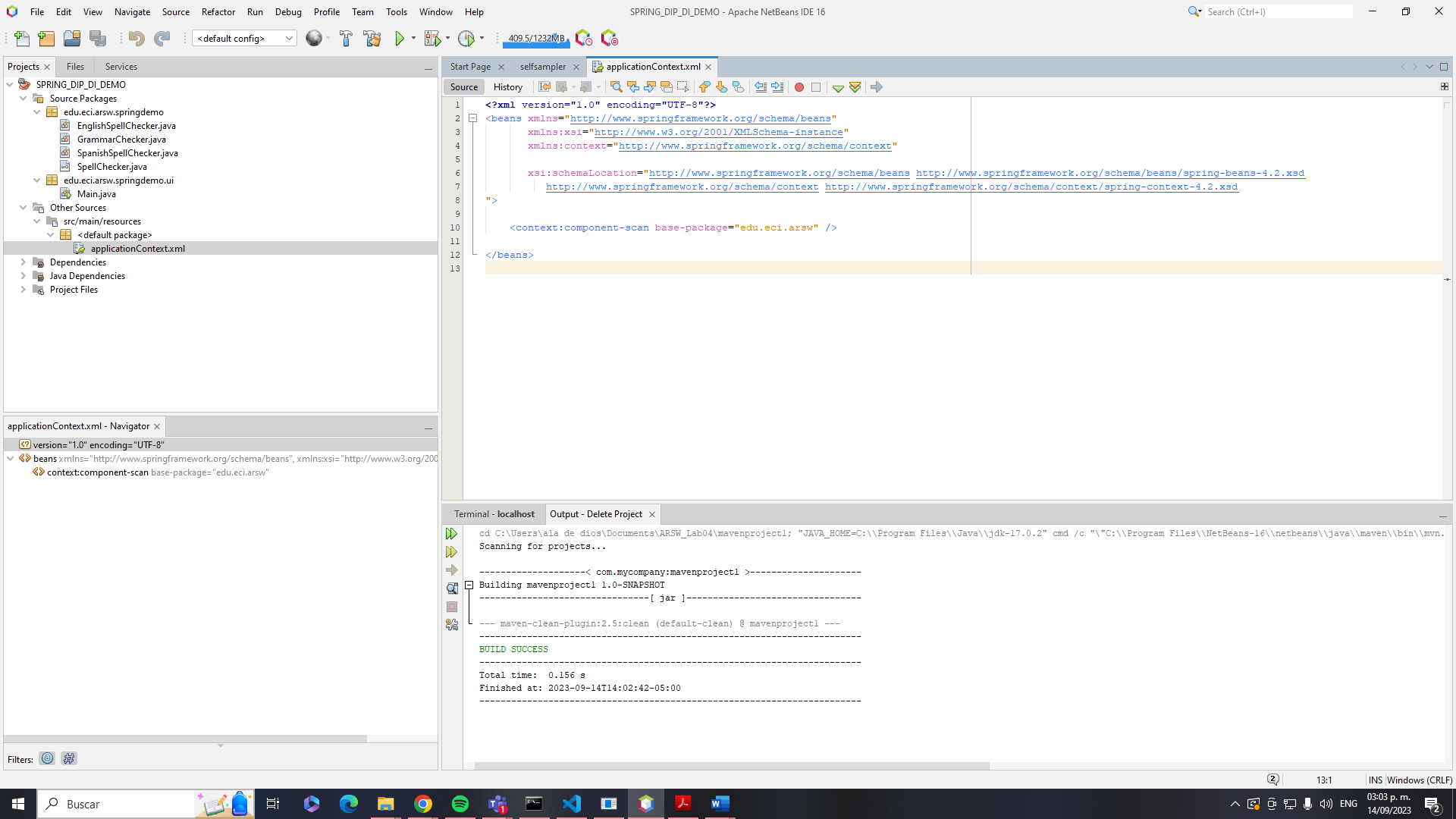
**Parte I. Ejercicio básico.**

Para ilustrar el uso del framework Spring, y el ambiente de desarrollo para el uso del mismo a través de Maven (y NetBeans), se hará la configuración de una aplicación de análisis de textos, que hace uso de un verificador gramatical que requiere de un corrector ortográfico. A dicho verificador gramatical se le inyectará, en tiempo de ejecución, el corrector ortográfico que se requiera (por ahora, hay dos disponibles: inglés y español).

1. Abra las fuentes del proyecto en NetBeans.
2. Revise el archivo de configuración de Spring ya incluido en el proyecto (src/main/resources). El mismo indica que Spring buscará automáticamente los 'Beans' disponibles en el paquete indicado.

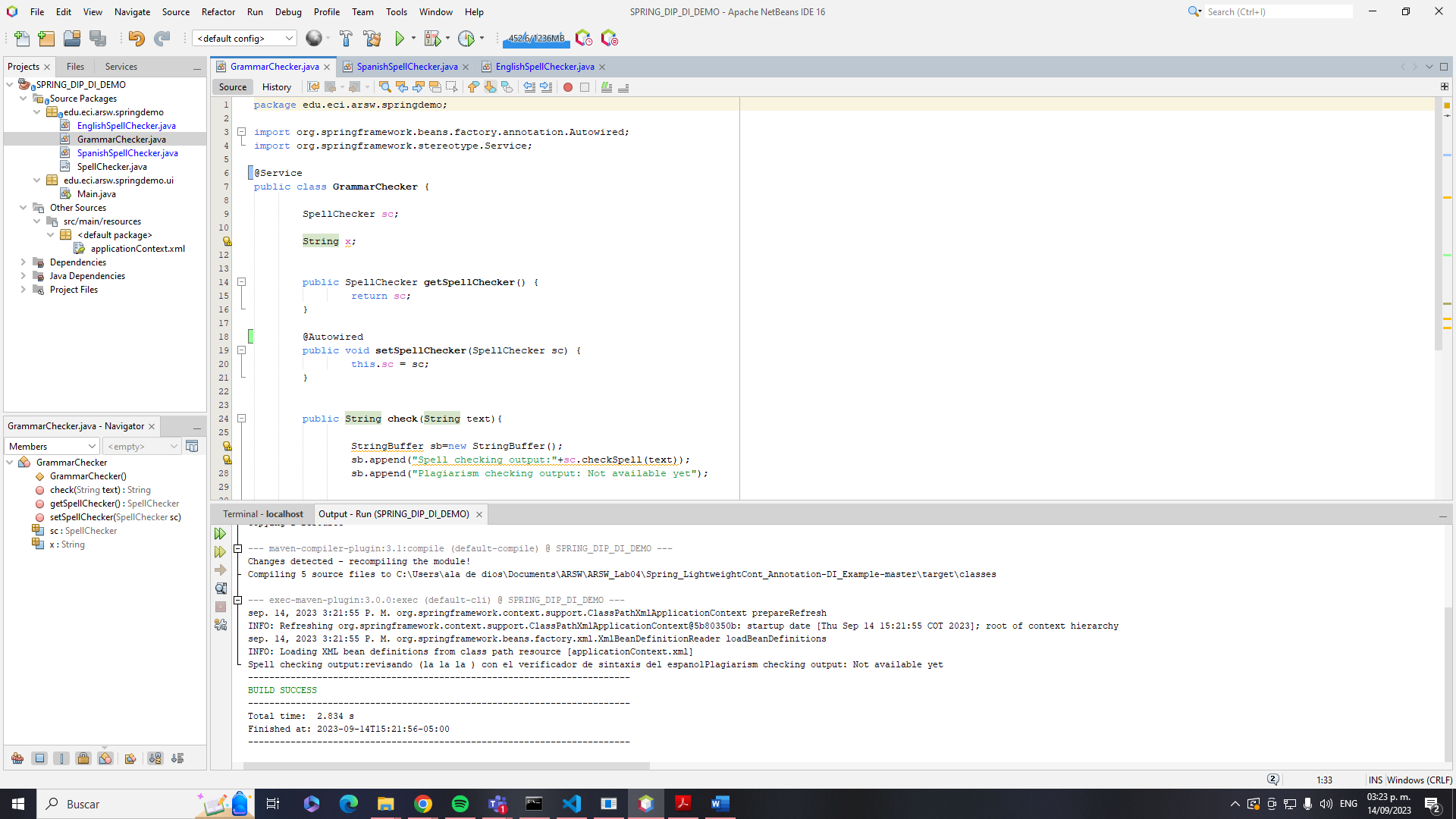
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente



1. Haciendo uso de la [configuración de Spring basada en anotaciones](https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-spring-beans-and-dependency-injection.html) marque con las anotaciones @Autowired y @Service las dependencias que deben inyectarse, y los 'beans' candidatos a ser inyectadas -respectivamente-:
   * GrammarChecker será un bean, que tiene como dependencia algo de tipo 'SpellChecker'.
   * EnglishSpellChecker y SpanishSpellChecker son los dos posibles candidatos a ser inyectados. Se debe seleccionar uno, u otro, más NO ambos (habría conflicto de resolución de dependencias). Por ahora haga que se use EnglishSpellChecker.

* Modificación de la clase GrammarChecker



* Modificación de la clase SpanishSpellChecker

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* Modificación de la clase EnglishSpellChecker

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para que no haya un conflicto de resolución de dependencias cometamos el @Service de la clase SpanishSpellChecker.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Haga un programa de prueba, donde se cree una instancia de GrammarChecker mediante Spring, y se haga uso de la misma:

Interfaz de usuario gráfica, Texto

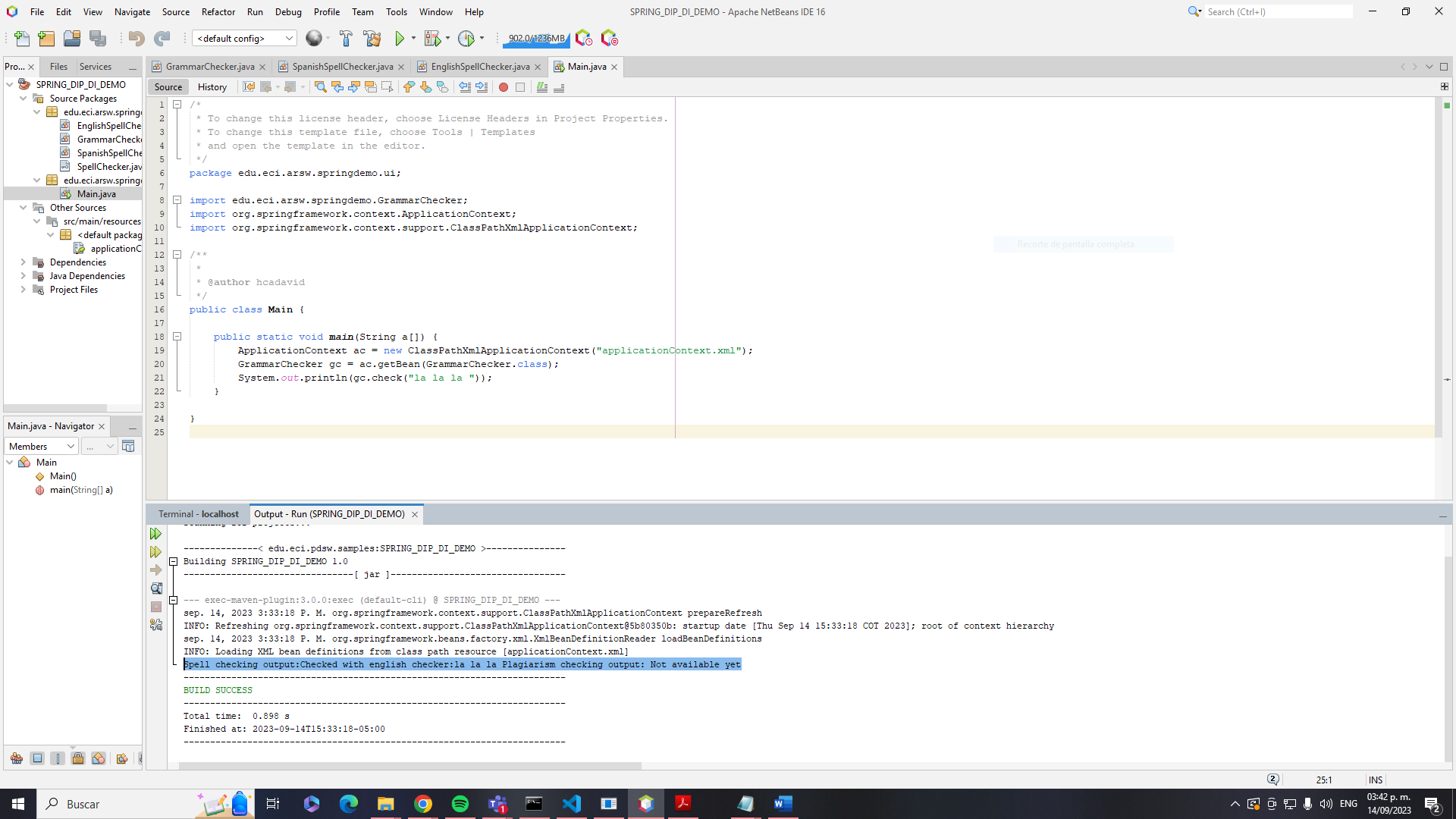
Descripción generada automáticamente

Como podemos observar la clase Main ya cuenta con el programa de prueba donde se crea la instancia de GrammarChecker mediante Spring

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

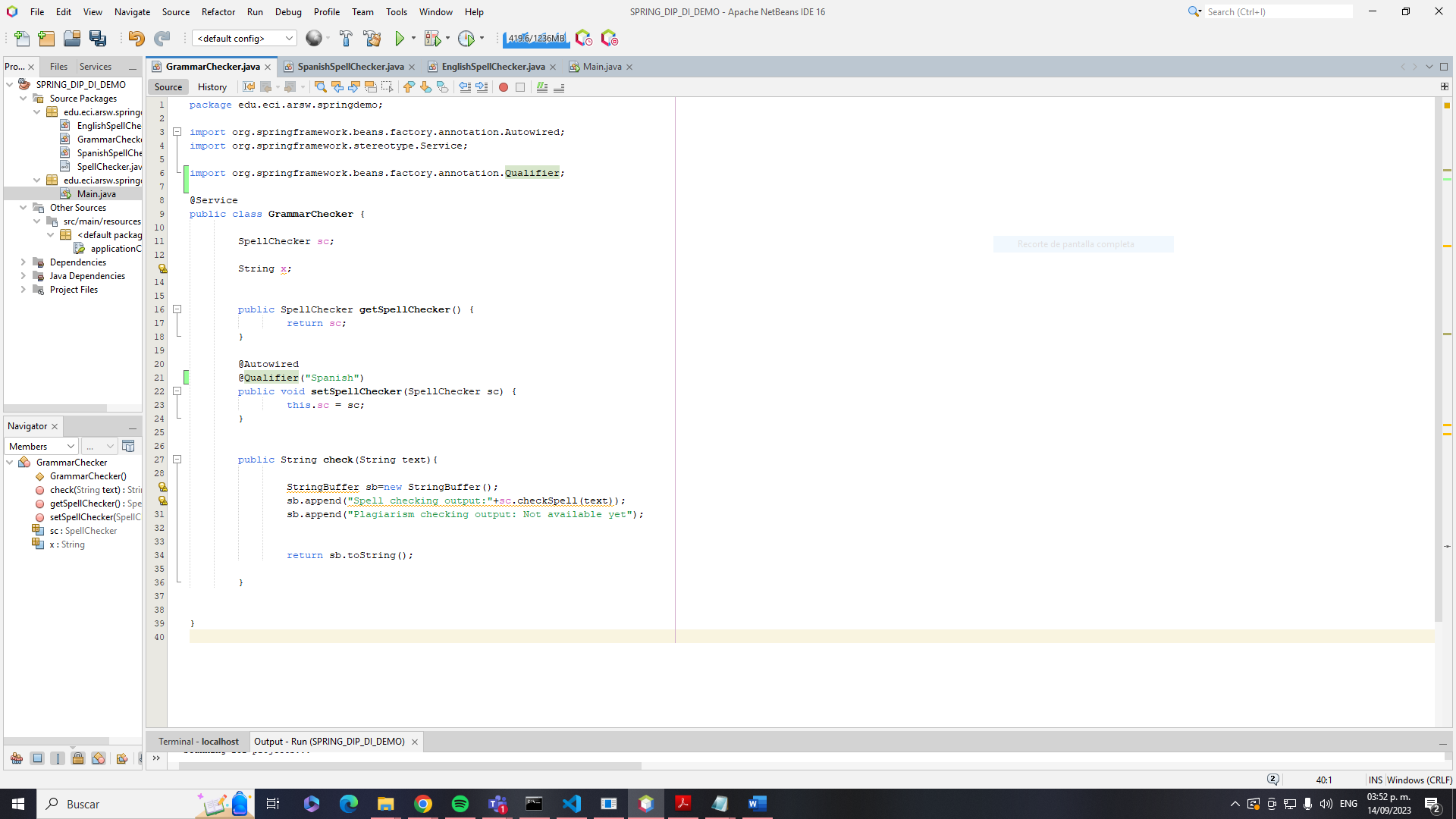
Descripción generada automáticamente

* Resultados de ejecución:



1. Modifique la configuración con anotaciones para que el Bean ‘GrammarChecker’ ahora haga uso de la clase SpanishSpellChecker (para que a GrammarChecker se le inyecte EnglishSpellChecker en lugar de SpanishSpellChecker. Verifique el nuevo resultado.

* Modificación de la clase GrammarChecker



* Modificación de la clase SpanishSpellChecker

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Modificación de la clase EnglishSpellChecker

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Resultados de ejecución:

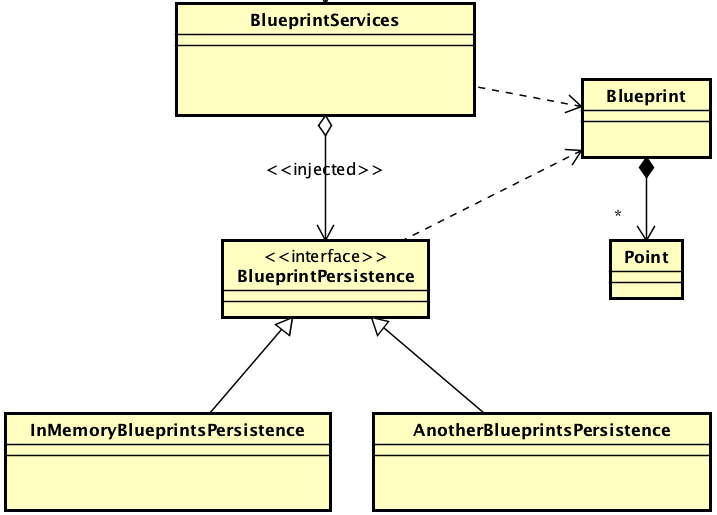
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Componentes y conectores - Parte I.**

**Middleware- gestión de planos.**

En este ejercicio se va a construir un modelo de clases para la capa lógica de una aplicación que permita gestionar planos arquitectónicos de una prestigiosa compañía de diseño.

[](https://github.com/ARSW-ECI-archive/SpringBoot_REST_API_Blueprints/blob/master/img/ClassDiagram1.png)

1. Configure la aplicación para que funcione bajo un esquema de inyección de dependencias, tal como se muestra en el diagrama anterior.

Lo anterior requiere:

* + Agregar las dependencias de Spring.
  + Agregar la configuración de Spring.
  + Configurar la aplicación -mediante anotaciones- para que el esquema de persistencia sea inyectado al momento de ser creado el bean 'BlueprintServices'.

1. Complete las operaciones getBluePrint() y getBlueprintsByAuthor(). Implemente todo lo requerido de las capas inferiores (por ahora, el esquema de persistencia disponible 'InMemoryBlueprintPersistence') agregando las pruebas correspondientes en 'InMemoryPersistenceTest'.
2. Haga un programa en el que cree (mediante Spring) una instancia de BlueprintServices, y rectifique la funcionalidad del mismo: registrar planos, consultar planos, registrar planos específicos, etc.
3. Se quiere que las operaciones de consulta de planos realicen un proceso de filtrado, antes de retornar los planos consultados. Dichos filtros lo que buscan es reducir el tamaño de los planos, removiendo datos redundantes o simplemente submuestrando, antes de retornarlos. Ajuste la aplicación (agregando las abstracciones e implementaciones que considere) para que a la clase BlueprintServices se le inyecte uno de dos posibles 'filtros' (o eventuales futuros filtros). No se contempla el uso de más de uno a la vez:
   * (A) Filtrado de redundancias: suprime del plano los puntos consecutivos que sean repetidos.
   * (B) Filtrado de submuestreo: suprime 1 de cada 2 puntos del plano, de manera intercalada.
4. Agregue las pruebas correspondientes a cada uno de estos filtros, y pruebe su funcionamiento en el programa de prueba, comprobando que sólo cambiando la posición de las anotaciones -sin cambiar nada más-, el programa retorne los planos filtrados de la manera (A) o de la manera (B).
5. **Conclusiones**