PROGRAMACIÓN III

Trabajo Práctico - Fundamentos de Spring Boot

OBJETIVO GENERAL

Aplicar los conceptos fundamentales de Spring Boot para crear una aplicación profesional que utilice inyección de dependencias, estereotipos, configuración mediante properties y gestión de diferentes entornos con profiles.

MARCO TEÓRICO

Concepto	Aplicación en el proyecto
Application Context	Contenedor de IoC que gestiona el ciclo de vida de los beans y sus dependencias.
Beans	Objetos gestionados por Spring que representan componentes de la aplicación.
Inyección de Dependencias	Permite desacoplar componentes mediante la inyección automática de dependencias, preferentemente por constructor.
Estereotipos	Anotaciones especializadas (@Service, @Repository, @Component) que definen el rol de cada clase en la arquitectura.
Configuración y Profiles	Uso de application.properties y profiles para configurar la aplicación según el entorno (dev, prod).

CASO PRÁCTICO: Sistema de Gestión de Tareas

Desarrollarás una aplicación Spring Boot completa que gestione tareas (To-Do List) aplicando buenas prácticas profesionales.

📌 **Importante:** Este es un único proyecto integrador. Sigue las partes en orden para construir la aplicación paso a paso. Las consignas te indican QUÉ hacer, pero deberás pensar CÓMO implementarlo aplicando lo aprendido.

PARTE 1: Crear el Proyecto Base 🚀

1. Usa **Spring Initializr** (start.spring.io) para crear un proyecto con:
   - Maven, Java 17+, Spring Boot 3.x
   - Group: `com.utn`, Artifact: `tareas`
   - Dependencias: **Spring Boot DevTools** y **Lombok** (recomendado)
2. Descarga, descomprime y abre el proyecto en tu IDE
3. Crea la estructura de paquetes profesional:
   - `com.utn.tareas.model`
   - `com.utn.tareas.repository`
   - `com.utn.tareas.service`
4. Ejecuta el proyecto y verifica que inicia sin errores

PARTE 2: Modelo y Repositorio 📦

2.1. Clase Tarea

En el paquete `model`, crea:

- **Enum** **`Prioridad`** con valores: `ALTA`, `MEDIA`, `BAJA`
- **Clase** **`Tarea`** con atributos:
  - `Long id`
  - `String descripcion`
  - `boolean completada`
  - `Prioridad prioridad`
- Agrega constructor, getters, setters y `toString()`

2.2. Clase TareaRepository

En el paquete `repository`, crea la clase `TareaRepository`:

- Anota con `@Repository`
- Declara un `List<Tarea>` para almacenar tareas en memoria
- En el **constructor**, inicializa la lista con 3-5 tareas de ejemplo
- Implementa métodos para:
  - Guardar una tarea (genera ID automático)
  - Obtener todas las tareas
  - Buscar por ID (retorna `Optional<Tarea>`)
  - Eliminar por ID

💡 **Ayuda:** Para generar IDs automáticos, usa un contador o `AtomicLong`.

PARTE 3: Servicio de Lógica de Negocio 💼

En el paquete `service`, crea la clase `TareaService`:

- Anota con `@Service`
- **Inyecta `TareaRepository` por constructor** (buena práctica)
- Implementa métodos para:
  - Agregar una nueva tarea (recibe descripción y prioridad)
  - Listar todas las tareas
  - Listar solo las tareas pendientes (no completadas)
  - Listar solo las tareas completadas
  - Marcar una tarea como completada (recibe ID)
  - Obtener estadísticas (retorna String formateado con: total, completadas, pendientes)

💡 **Recuerda:** Todos los métodos deben usar el repositorio inyectado para acceder a los datos.

PARTE 4: Configuración con Properties ⚙️

En `src/main/resources/application.properties`, define:

- `app.nombre` - Nombre de tu aplicación
- `app.max-tareas` - Límite máximo de tareas permitidas
- `app.mostrar-estadisticas` - Si se muestran estadísticas (true/false)

Modifica `TareaService`:

- Inyecta las propiedades usando `@Value("${...}")`
- En el método de agregar tarea, valida que no se supere `max-tareas`
- Crea un método que imprima las propiedades de configuración

PARTE 5: Profiles para Diferentes Entornos 🔧

5.1. Configurar archivos de properties

- En `application.properties`, agrega: `spring.profiles.active=dev`
- Crea `application-dev.properties` con valores para desarrollo:
  - `app.max-tareas=10`
  - `app.mostrar-estadisticas=true`
  - `logging.level.com.utn.tareas=DEBUG`
- Crea `application-prod.properties` con valores para producción:
  - `app.max-tareas=1000`
  - `app.mostrar-estadisticas=false`
  - `logging.level.com.utn.tareas=ERROR`

5.2. Beans condicionales

En el paquete `service`:

- Crea la **interfaz** `MensajeService` con dos métodos:
  - `mostrarBienvenida()`
  - `mostrarDespedida()`
- Crea `MensajeDevService`:
  - Implementa `MensajeService`
  - Anota con `@Service` y `@Profile("dev")`
  - Mensajes detallados y amigables para desarrollo
- Crea `MensajeProdService`:
  - Implementa `MensajeService`
  - Anota con `@Service` y `@Profile("prod")`
  - Mensajes simples y concisos para producción

💡 **Concepto clave:** Solo se creará UNO de estos beans según el profile activo. Esto es una práctica profesional para adaptar el comportamiento según el entorno.

PARTE 6: Clase Principal y Ejecución ▶️

Modifica `TareasApplication` para que implemente `CommandLineRunner`:

- Inyecta `TareaService` y `MensajeService` **por constructor**
- En el método `run(...)`, implementa el siguiente flujo:
  1. Mostrar mensaje de bienvenida (usando `MensajeService`)
  2. Mostrar la configuración actual
  3. Listar todas las tareas iniciales
  4. Agregar una nueva tarea
  5. Listar tareas pendientes
  6. Marcar una tarea como completada
  7. Mostrar estadísticas
  8. Listar tareas completadas
  9. Mostrar mensaje de despedida

Pruebas:

1. Ejecuta la aplicación con `spring.profiles.active=dev`
2. Cambia a `spring.profiles.active=prod` y ejecuta nuevamente
3. Observa las diferencias en mensajes, límites y comportamiento
4. Toma capturas de pantalla de ambas ejecuciones

📚 Conclusiones Esperadas

- ✅ Crear proyectos Spring Boot con estructura profesional
- ✅ Aplicar inyección de dependencias por constructor
- ✅ Usar estereotipos según la responsabilidad de cada clase
- ✅ Configurar aplicaciones con properties
- ✅ Inyectar valores de configuración con `@Value`

- ✅ Gestionar diferentes entornos con profiles
- ✅ Crear beans condicionales con `@Profile`
- ✅ Implementar `CommandLineRunner` para lógica de inicio

📦 Entrega del Trabajo Práctico

Crea un **repositorio público en GitHub** con tu proyecto. El repositorio debe incluir:

1. **Código completo y funcional** con la estructura de paquetes correcta
2. **Archivos de configuración:**
   - `application.properties`
   - `application-dev.properties`
   - `application-prod.properties`
3. **README.md profesional** que contenga:
   - 📝 Descripción del proyecto
   - ⚙️ Tecnologías utilizadas (Java, Spring Boot, Maven)
   - 🚀 Instrucciones para clonar y ejecutar el proyecto
   - 🔧 Cómo cambiar entre profiles (dev/prod)
   - 📸 Capturas de pantalla de la consola con ambos profiles
   - 💭 Conclusiones personales sobre lo aprendido
   - 👤 Tu nombre y legajo
4. **Commits significativos** que muestren el progreso del desarrollo

🔗 **Formato de entrega:** Envía el link de tu repositorio GitHub en la tarea de Moodle.

💡 **Tip profesional:** Un README bien documentado demuestra profesionalismo. Imagina que otro desarrollador debe entender tu proyecto solo leyendo el README.