ACM GiWeb - Java Backend Spring

Talleres — Comportamientos del IoC en Spring Framework

Introducción

El contenedor de inversión de control (IoC container) de Spring administra los objetos (beans) de una aplicación, gestionando su creación, ciclo de vida, dependencias y visibilidad. Sin embargo, los beans no siempre se comportan como se espera: sus interacciones pueden generar resultados inesperados si no comprendemos cómo opera internamente el contenedor. A través de los siguientes talleres, pondrán a prueba tu capacidad de análisis, observación y resolución de comportamientos anómalos, comprendiendo la lógica real detrás del motor de Spring.

Objetivos de Aprendizaje

- Comprender cómo se crean, gestionan y destruyen los beans en el contenedor IoC.
- Diferenciar los efectos de las anotaciones @Component, @Service, @Repository, @Configuration y @Bean.
- Analizar los efectos de los scopes (singleton, prototype, request, session).
- Resolver conflictos de inyección y dependencias circulares.
- Explicar comportamientos inesperados como lazy loading inefectivo o duplicación de beans.
- Documentar hallazgos y soluciones con pensamiento crítico.

Criterios de Evaluación

Criterio	Descripción	Puntaje
Análisis técnico	Identifica correctamente el comportamiento inesperado y sus causas.	10 pts
Implementación	Reproduce los escenarios propuestos y demuestra comprensión del	15 pts

	contenedor.	
Solución y justificación	Propone y explica soluciones adecuadas (código, configuración o rediseño).	15 pts
Documentación y presentación	Presenta README claro, evidencia de ejecución y reflexión final.	10 pts

Punto 1 — El Laboratorio de los Beans

Tema: Creación, inyección y ciclo de vida de beans.

Escenario: Tu equipo fue asignado a un microproyecto llamado BeanLab, donde se realizan experimentos con la creación y destrucción de objetos dentro del contenedor IoC.

Retos:

1. Crea una clase ExperimentService anotada con @Component que imprima mensajes en su metodo constructor.
2. Declara otro bean manualmente con @Bean dentro de una clase @Configuration, puedes nombrar los beans con @Component(“nombreBean”) y @Bean(“nombreBean”) para posteriormente utilizarlos con @Qualifier
3. Agrega @Lazy a uno y observa cuándo se ejecuta su inicialización
4. Intercambia las anotaciones de @Lazy entre cada caso y observa su comportamiento.

Producto esperado:

Un breve reporte donde describas el ciclo de vida de cada bean y la diferencia entre los creados manualmente.

Punto 2 — Los Clones del Contenedor
Tema: Scopes y gestión de instancias.

Escenario: Estás implementando un sistema de carrito de compras. Los usuarios reportan que todos los carritos comparten los mismos productos.

Retos:

1. Crea un bean ShoppingCart con @Scope('prototype') y otro ProductService singleton que lo use.Demuestra que el carrito es el mismo para todos los usuarios.
2. Corrige el comportamiento usando ObjectProvider, inyectando el ApplicationContext desde el singleton y con applicationContext.getBean() o @Lookup.
3. Simula varios usuarios con ejemplos en consola.
4. Cambia el scope a singleton, ¿Es también el mismo carrito de compras para todos los usuarios?

Producto esperado:
Explicación del comportamiento de los scopes, diferencias entre singleton y prototype, y evidencia de las instancias distintas.

Punto 3 — La Conspiración de los Qualifiers
Tema: Resolución de dependencias y ambigüedades.

Escenario: La tienda tiene dos estrategias de descuento: básica y premium, pero el contenedor no parece decidir correctamente cuál usar.

Retos:

1. Crea dos implementaciones de la interface DiscountService: BasicDiscountService y PremiumDiscountService.
2. Inyéctalas en OrderService sin especificar cuál usar. Observa el error.
3. Usa @Primary en una de ellas y verifica el resultado.
4. Sustituye con @Qualifier y cambia el nombre del bean para provocar confusión.
5. Usa @Autowired(required=false) en un bean opcional y observa el comportamiento cuando no existe en el contexto.

Producto esperado:
Reporte con ejemplos de errores de inyección y soluciones aplicadas. Explica cuándo usar @Primary, @Qualifier y required=false.

Punto 4 — El Bucle Infinito

Tema: Dependencias circulares, orden de inicialización y contexto de carga.

Escenario: InventoryService depende de OrderService para verificar ventas, y OrderService depende de InventoryService para validar stock. El sistema no inicia.

Retos:

1. Crea ambos servicios con inyección por constructor y observa el error.
2. Cambia una de las dependencias a inyección por setter o @Lazy.
3. Analiza si la app arranca y qué logs aparecen.
4. Identifica el orden de inicialización de los beans en consola.
5. Propón una solución de diseño (refactorización, otro tipo de inyección, otros).

Producto esperado:

Explicación técnica de por qué ocurre la excepción BeanCurrentlyInCreationException o UnsatisfiedDependencyException, y cómo evitar dependencias circulares en el diseño.