1. ¿Qué es la inyección de dependencia y representa los tipos de inyección de dependencia que existen?

El principio de inyección de dependencias no es más que pasar las dependencias cuando sea necesario en lugar de inicializar las dependencias dentro de la clase receptora. Este principio permite desacoplar la construcción de sus clases de la construcción de las dependencias de sus clases, es decir, que por un lado se crearía la clase y por otro lado las instancias de esta clase.

Utilizando el framework de Spring nos permite hacer esta inyección de dependecias. Spring llama a estas instancias beans, es decir, los beans son las instancias de las clases que van a estar disponibles para ser reutilizadas y son gestionadas por el contenedor de Spring.

Los tipos de inyección de dependencias que existen son los setters y constructor.

1. Menciona al menos 5 patrones de diseño y para que los utilizas.

Singlenton: se utiliza cuando se quiere crear una sola instancia de una clase. Para ello el constructor de esa clase se define como privado.

Decorator o decorador: permite agregar funcionalidades a un objeto sin alterar el comportamiento de otras instancias de objetos de la misma clase. Usando este patrón de diseño es posible agregar o quitar responsabilidades a un objeto en el runtime.

Mediator o mediador: define un objeto el cual encapsula como otros objetos van a interactuar. Es considerador un patrón de comportamiento porque puede alterar cómo se ejecuta la aplicación.

Adapter o adaptador: permite que un objeto que no está diseñado para funcionar con una interfaz sea modificado para que este sí pueda funcionar con esta.

MVC (Modelo Vista Controlador): se utiliza para desarrollar interfaces de usuario, dividiendo la lógica entre elementos interconectados: modelo, vista, controlador. El modelo contiene la lógica de la aplicación, se encarga de llamar a los distintos datos y contiene las reglas de negocios. La vista es la representación visual de los datos que se recolectan. Y el controlador se encarga de recibir los datos, transformarlos de forma tal que sean útiles para el modelo, recibe los datos del modelo y luego construye la respuesta para el usuario.

1. Representa y explica cada uno de los componentes de la arquitectura del framework Spring.

Spring Context: para la inyección de dependencias.

Spring POA: para la Programación Orientada a Aspectos(POA).

Spring DAO: para las operaciones con bases de datos usando el patrón DAO (Data Access Object).

Spring JDBC: soporta tanto JDBC (Java DataBase Conectivity) como DataSource.

Spring ORM: para herramientas ORM (Object Relational Mapping) como lo es Hibernate.

Spring Web: para la creación de aplicaciones web.

Spring MVC: para las implementaciones MVC que permiten crear aplicaciones web, servicios webs, etc.

1. Describe para que se utilizan las siguientes anotaciones

@Autowired: es una anotación utilizada para automatizar la inyección de dependencias.

@Qualifier: esta anotación es utilizada cuando se tienen varias implementaciones de una misma interfaz, de forma tal que se le dice a Spring cuál dependencia debe inyectar en esa clase y evitar confusiones.

@PostConstruct: es una anotación que precede al método que se ejecuta justo después de haberse creado el bean.

@PreDestroy: lo opuesto de @PostConstruct, precede al método que se ejecuta justo antes de destruirse el bean.

@Required: indica que un atributo es requerido para que el bean pueda ser definido correctamente.

@Bean: indica que el bean va a ser manejado por el contenedor de Spring.

@Configuration: esta anotación le dice a Spring que cierta clase va a ser una clase de configuración.

@Transactional:

@Secured: identifica la lista de roles de un método. Un usuario solo puede acceder a un método que tenga esta anotación si tiene definido al menos unos de los roles especificados.

@RollesAllowed: muy parecido al @Secure.

@ManagedResource: indica que la clase anotada es un recurso que será gestionado con JMX (Java Managed eXtensions)

@ManageAttribute:

@ManageOperation:

@Cacheable: es una anotación utilizada en los métodos para revisar si ya se cuenta con una respuesta en el caché con los mismos parámetros (misma petición) antes de invocar el método en sí. En caso contrario, se hace la llamada al método normalmente y la respuesta es guardar en caché para futuras invocaciones con los mismos parámetros.

1. Para qué sirve un converter.

Es una interfaz descrita en Java que puede manejar la conversión Objeto a String y String a Objeto, entre objetos de modelos de datos a representaciones de String de esos objetos.

1. ¿Qué es un aspecto y describe un ejemplo de cómo lo podríamos implementar?

Un aspecto es una funcionalidad transversal encapsulada que puede ser ejecutada antes o después de invocar una clase. Varias clases pueden utilizar el mismo aspecto.

Ejemplo: Tenemos una clase cliente que tiene el método “insertarCliente ()” en la base de datos, pero antes es necesario saber si el usuario que está invocando al método tiene los privilegios para poder insertar un cliente en la base de datos, es decir, que esté logueado y autenticado. Para resolver este problema lo mejor sería aplicar un aspecto que contenga las funcionalidades de logueo y autenticado y mediante un archivo de configuraciones definir su uso para el método “insertarCliente ()”. Lo que se logra con esto, es que cada vez que se invoque el método “insertarCliente ()”, se ejecutará el aspecto y verificará sí efectivamente el usuario que está haciendo la invocación está logueado y autenticado.

1. La empresa XYZ es la encargada de administrar el reporte de incidencias de los equipos de cómputo, donde un operador puede dar de alta una incidencia (“definir los datos que consideres más importante”), consultar incidencias por ID, por día, y por operador que las dio de alta, cabe mencionar que debemos tener dashboards que representen número de peticiones a los servicios consultados, insertados, cuantos se resolvieron, cuantos están abiertos.
2. ¿Qué es la arquitectura de microservicios?

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de esos servicios es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. Es una arquitectura que vino a sustituir los monolitos.

1. Menciona para que sirve cada uno de los siguientes componentes
2. Spring Cloud Config: herramienta para externalizar la configuración de las aplicaciones, generalmente se utiliza en sistemas distribuidos, como los microservicios, para gestionar de forma centralizada estas configuraciones.

b. Eureka: es un servidor para el registro y localización de microservicios, balanceo de carga y tolerancia de fallos.

c. Ribbon: librería encargada de realizar el balanceo de carga.

d. Hystrix:

e. Zuul:

f. OAUTH: protocolo de autorización que permite a terceros (clientes) acceder a contenidos de propiedad de un servidor sin que el servidor tenga que manejar o conocer las credenciales del usuario.

g. SWAGGER: serie de reglas, especificaciones y herramientas que ayudan a documentar las APIs.

h. SPLUNK:

1. Generar una tarea ya sea con MAVEN o GRADLE para generar las clases de un WSDL del mismo.