pool.setInitialSize(3);

- Este método establece el tamaño inicial del pool de conexiones. En este caso, se configura el tamaño inicial del pool a 3 conexiones. Esto significa que cuando se inicia la aplicación, se crearán y se mantendrán abiertas 3 conexiones a la base de datos o al recurso compartido especificado en el pool, para que estén listas para su uso inmediato.

pool.setMinIdle(3);

- Este método establece el número mínimo de conexiones inactivas que el pool debe mantener en todo momento. En este caso, se configura el número mínimo de conexiones inactivas en 3. Esto asegura que, incluso si no se están utilizando activamente todas las conexiones disponibles, siempre habrá al menos 3 conexiones listas y disponibles para su uso. Esto ayuda a reducir la latencia cuando se necesita una conexión, ya que no es necesario crear una nueva conexión desde cero cada vez.

pool.setMaxIdle(8);

- Este método establece el número máximo de conexiones inactivas que el pool puede mantener a la vez. En este caso, se configura el número máximo de conexiones inactivas en 8. Esto significa que, aunque se haya establecido un mínimo de 3 conexiones inactivas, el pool no permitirá que haya más de 8 conexiones inactivas al mismo tiempo. Si hay más conexiones inactivas de las necesarias, el pool puede cerrar las conexiones adicionales para ahorrar recursos

@Dependent:

* @Dependent es el alcance predeterminado en CDI. Cada vez que se inyecta un componente anotado con @Dependent, se crea una nueva instancia de ese componente. Esto significa que el ciclo de vida de la instancia depende del componente que la inyecta. Se puede utilizar @Dependent cuando se necesita una instancia completamente independiente en cada inyección, sin compartir estado entre ellas.

@RequestScoped:

* @RequestScoped es un alcance de ciclo de vida que mantiene la instancia del componente durante una sola solicitud HTTP. Cada solicitud obtiene una nueva instancia de los componentes anotados con este alcance. Útil para componentes que deben mantener estado solo durante el procesamiento de una solicitud web, como los controladores de solicitudes web.

@SessionScoped:

* @SessionScoped es un alcance de ciclo de vida que mantiene la instancia del componente durante la duración de la sesión del usuario. Cada usuario tiene su propia instancia de los componentes anotados con este alcance. Se usa para componentes que deben mantener estado a lo largo de múltiples solicitudes durante la misma sesión de un usuario, como la información de inicio de sesión o una cesta de compras en una aplicación web de comercio electrónico.

@ConversationScoped:

* @ConversationScoped es un alcance de ciclo de vida que mantiene la instancia del componente durante una conversación específica. Una conversación puede involucrar varias solicitudes y respuestas del usuario. Se inicia y se detiene manualmente. Útil cuando se necesita mantener el estado a lo largo de una secuencia de interacciones en la aplicación, como un flujo de compra en línea, donde se desean guardar datos en varias páginas durante una única conversación.

@ApplicationScoped:

* @ApplicationScoped es un alcance de ciclo de vida que mantiene una única instancia del componente durante toda la duración de la aplicación. Esta instancia es compartida por todos los usuarios de la aplicación. Se utiliza para componentes que deben mantener estado global o configuraciones compartidas en toda la aplicación, como una caché de datos o una configuración de aplicación.

@Retention

se utiliza para controlar cuándo las anotaciones estarán disponibles y cuándo serán visibles para el código en tiempo de ejecución, lo que es crucial para la configuración y personalización

@Target

se utiliza para especificar en qué elementos del código una anotación personalizada puede ser aplicada. Esto define el alcance o el contexto en el que la anotación puede utilizarse, como clases, métodos, campos, parámetros, constructores, variables locales y paquetes. como por ejemplo si se establece @Target(ElementType.METHOD).

@Qualifier

se utiliza para resolver ambigüedades en la inyección de dependencias cuando hay múltiples implementaciones disponibles de un mismo tipo. Permite etiquetar implementaciones específicas con nombres calificadores y luego seleccionarlas de manera precisa al realizar inyecciones de dependencias.

sendRedirect:

sendRedirect es un método de la interfaz HttpServletResponse.

Se utiliza para enviar una respuesta de redirección al cliente (navegador).

Cuando se invoca sendRedirect, el navegador del cliente recibe un código de estado HTTP 302 (redirección temporal) y se le indica que solicite la nueva URL especificada.

Este método se puede usar para redirigir a cualquier recurso, incluso fuera del contexto de la aplicación web actual.

Por lo general, se utiliza para redirigir a páginas fuera del contexto de la aplicación o cuando se requiere una redirección externa.

getRequestDispatcher:

getRequestDispatcher es un método de la interfaz ServletRequest.

Se utiliza para obtener un objeto RequestDispatcher, que se utiliza para enviar la solicitud a otro recurso, como un servlet, JSP o cualquier otro recurso en el contexto de la aplicación.

Este método se utiliza principalmente para redireccionar dentro del contexto de la aplicación, lo que significa que se mantiene la URL original en el navegador del cliente.

El recurso al que se redirige puede ser dinámico (como un servlet o JSP) o estático (como un archivo HTML).